176 #4

169.1595

PATENT APPLICATION

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:	)	
	:	Examiner: Unassigned
ALISON JOAN LENNON	)	
	:	Group Art Unit: Unassigned
Application No.: 09/493,220	)	
	:	
Filed: January 28, 2000	)	
	:	
For: BROWSING ELECTRONICALLY-ACCESSIBLE RESOURCES	) :	March 28, 2000

Assistant Commissioner for Patents
Washington, D.C. 20231

CLAIM TO PRIORITY

Sir:

Applicant hereby claims priority under the International Convention and all rights to which she is entitled under 35 U.S.C. § 119 based upon the following Australian Priority Applications:

PP 8374	Australia	January 29, 1999;
PP 8375	Australia	January 29, 1999;
PP 8376	Australia	January 29, 1999;
PP 8377	Australia	January 29, 1999; an
PQ 4612	Australia	December 13, 1999

DC_MAIN 19898 v 1

A certified copy of each of the priority documents is enclosed.

Applicant's undersigned attorney may be reached in our Washington, D.C. office by telephone at (202) 530-1010. All correspondence should continue to be directed to our address given below.

Respectfully submitted,

Attorney for Applicant
Lawrence A. Stahl
Registration No. 30,110

FITZPATRICK, CELLA, HARPER & SCINTO
30 Rockefeller Plaza
New York, New York 10112-3801
Facsimile: (212) 218-2200

LAS:SWF:eyw

169.1595

RECEIVED
JUL 18 2000
GROUP 270

OIPE JC89
MAR 28 2000
PATENT & TRADEMARK OFFICE

RECEIVED
MAR 28 2001
Technology Center 2100

Patent Office
Canberra

RECEIVED
TC 1700 MAIL ROOM

I, ANNA MAIJA MADL, ACTING TEAM LEADER EXAMINATION SUPPORT & SALES hereby certify that annexed is a true copy of the Provisional specification in connection with Application No. PP 8374 for a patent by CANON KABUSHIKI KAISHA filed on 29 January 1999.

I further certify that pursuant to the provisions of Section 38(1) of the Patents Act 1990 a complete specification was filed on 28 January 2000 and it is an associated application to Provisional Application No. PP 8374 and has been allocated No. 13613/00.

WITNESS my hand this
Ninth day of February 2000

RECEIVED
MAR 29 2000
TC 1700 MAIL ROOM

COMMONWEALTH OF AUSTRALIA
AUSTRALIA
PATENT OFFICE

A.M. Madl.

ANNA MAIJA MADL
ACTING TEAM LEADER
EXAMINATION SUPPORT & SALES

Appln. No.: 09/493,220
Filed: January 28, 2000
Inv.: Alison Joan Lennon
Title: Browsing Electronically Accessible Resources

CERTIFIED COPY OF
PRIORITY DOCUMENT

S & F Ref: 447165

ORIGINAL

AUSTRALIA

Patents Act 1990

PROVISIONAL SPECIFICATION FOR THE INVENTION ENTITLED:

Method and Apparatus for Generating Printed Presentation(s)

Name and Address of Applicant: Canon Kabushiki Kaisha, incorporated in Japan, of 30-2, Shimomaruko 3-chome, Ohta-ku, Tokyo, 146, JAPAN

Name(s) of Inventor(s): Alison Joan Lennon

This invention is best described in the following statement:

Method and Apparatus for generating printed presentation(s)

Copyright Notice

Field of Invention

The present invention relates to a method and apparatus for generating a printed presentation. The invention also relates to an apparatus and a computer program product for implementing the method.

Background

As network connectivity has continued its explosive growth and digital storage has become smaller, faster, and less expensive, the quantity of electronically-accessible resources has increased enormously. So much so that the discovery and location of the available resources has become a critical problem. These electronically-accessible resources can be digital content (e.g., digital images, video and audio) which may be available over the network, web-based resources (e.g., HTML/XML documents) and electronic devices (e.g., printers, displays, etc.). In addition, there are electronically-accessible catalogues of other resources, which may not be electronically accessible (e.g., books, analog film media, etc.). What is needed is a consistent method of describing resources so that location of resources, electronically-accessible or otherwise, can be more readily achieved.

The problems of consistent resource description are twofold. First, there is the problem of acceptance of a standard (consistent) method of resource description. The

second problem is related to the generation of descriptions. Often the cost of this process is significant.

If resources can be consistently described then it is possible to develop methods and build devices that facilitate resource discovery, understanding and presentation. Resources can be composite items involving other resources and can include schedules for presentation, delivery and/or consumption. Printed presentations can therefore be viewed as just another electronically-accessible resource and as such can be described in a similar way to items of digital content (like images, video and audio).

Summary of the Invention

It is an object of the present invention to ameliorate one or more disadvantages of the prior art.

One or more exemplary aspects of the invention are listed below, but are not limited thereto.

According to a first aspect of the invention, there is provided a method of generating a printed presentation based on a description of the digital resources, contained on a medium accessed by a source processing device, on a destination printing device, said method comprising the steps of: providing a description scheme for the class of presentations using a declarative description definition language which contains definitions for descriptor components of the description scheme, wherein each said descriptor component comprises the association of a resource attribute with a representative value for that attribute; associating with the said description scheme a set of presentation rules which specify characteristics of the style of the presentation for descriptions created using the said description scheme; creating a description of the said printed presentation using the said description scheme as the template for the description's instantiation in the source processing device; generating the printed presentation from the said description and any associated image content on the said destination printing device.

According to another aspect of the invention, there is provided an apparatus for implementing any one of the aforementioned methods.

According to another aspect of the invention there is provided a computer program product including a computer readable medium having recorded thereon a computer program for implementing any one of the methods described above.

Brief Description of the Drawings

Embodiments of the invention are described with reference to the drawings, in which:

Fig. 1A shows a flow diagram of the preferred method of generating a description of a resource in accordance with a preferred embodiment;

Fig. 1B shows a flow diagram of the preferred method of processing a description of a resource in accordance with a preferred embodiment;

Figs. 2A shows a flow diagram of a method of generating a typical document object model;

Fig 2B show a flow diagram of a method of generating a Description Object Model in accordance with a preferred embodiment;

Fig. 3 shows a UML class diagram showing core elements of the Dynamic Description Framework(DDF) data model in accordance with a preferred embodiment;

Fig. 4 shows a schematic drawing depicting the processing model of an exemplary description according to the preferred DDF;

Fig. 5 shows a schematic drawing depicting the processing model of another exemplary description according to the preferred DDF;

Fig. 6 shows a schematic drawing depicting the relationship between a description scheme (Document Type Definition) and descriptions(XML documents) in accordance with a preferred embodiment;

Fig. 7A is a flow diagram of a method of generating a description of a resource in accordance with a preferred embodiment;

Fig. 7B is a flow diagram of a method of processing a description of a resource in accordance with a preferred embodiment;

Fig. 8 is a flow diagram of a preferred method of extending a description of a resource;

Fig. 9 is a flow diagram of a preferred method of visually presenting a description of a resource;

Fig. 10 is a flow diagram of a preferred method of selecting one or more descriptions or part of one or more descriptions of a resource;

Fig. 11, is a flow diagram of a preferred method of translating a description of a resource;

Fig. 12 shows a Digital Video Browser System in accordance with a preferred embodiment;

Fig. 13 shows an implementation of the Digital Video Browser System in a remote handheld device in accordance with a preferred embodiment;

Fig. 14 shows an alternative implementation of the Digital Video Browser System in a remote handheld device in accordance with a preferred embodiment; and

Fig. 15 is a block diagram of a general-purpose computer for implementating the preferred methods shown in Figs 1A,1B,2B and 7A to 11.

Detailed Description

Where reference is made in any one or more of the accompanying drawings to steps and/or features, which have the same reference numerals, those steps and/or features have for the purposes of this description the same function(s) or operation(s), unless the contrary intention appears.

1.0 Overview of Preferred Method(s)

The preferred methods described herein are specific examples of a generalised form of a method for generating and processing descriptions of resources utilising a Dynamic Description Framework. This framework provides a data model, an application programming interface (API) and a serialisation syntax for use in the description of content, in particular audiovisual resources. The preferred DDF incorporates the benefits of declarative description of content with procedural methods for the creation and processing of descriptions and components of descriptions.

Fig. 1A shows an overview of a preferred method of generating a description of an electronically accessible resource. In this method, a description scheme 100A is read by a description generator 106A which in turn generates a description object model (DesOM) 108A in memory. The DesOM 108A represents the description in memory which can be serialised as an XML document for the purposes of storage and transport.. Preferably, both the description scheme 100A and description 110A are textual and are both readable by machines and humans. It is further preferable that the description scheme 100A is provided with DescriptorHandler(s) so as to provide operations/processes which can unambiguously provide descriptive information or other actions on the resource 104A. For example, the preferred method in one operating mode is able to automatically

generate a description 110A of the resource 104A. In this operating mode, the processes of the DescriptorHandler(s) operate on the resource 104A to generate a description 110A of that particular resource 104A. These description schemes 102 and descriptions 106 are defined in terms of the abovementioned Dynamic Description Framework (DDF).

Fig. 1B shows an overview of the preferred method of processing a description of an electronically accessible resource. In this method, a description 100B is parsed by a processor 102B which in turn generates a description object model (DesOM) 104B in memory. Such a description 100B may be generated in accordance with the method of Fig. 1A. Preferably, the processor 102B and description generator 106A (Fig. 1) are incorporated as one unit. The description 100B refers to a description scheme 106B which may in turn refer to a number of DescriptionHandler(s) 108B. The description 100B also refers to the resource 110B which the description describes. In this method, a set of rules may be applied to the DesOM 104B to generate a modified DesOM 112B which can be serialised as an XML document. These set of rules are defined by the Description Scheme 106B. The DescriptorHandlers 108B provide further processing of the DesOM 114B or DesOM+ 116B which in turn may be serialised as an XML document. In one operating mode, the preferred method is able to compute the similarity between content of the resources 110B. In the later mode, the DescriptorHandler provides a process for computing similarity between content of resources. The preferred method is further adapted to apply a set of rules 118B to the DesOM 104B. The set of rules 118B provides one or more associated actions on the DesOM 104B. The resultant output of these actions is itself a DesOM 112B. Further, a description scheme may be read into memory and a set of rules provided for performing one or more associated actions on the description scheme itself. The method by these sets of rules is able extend resource descriptions; translate resource descriptions; select one or more specific descriptions according to a query; visually present resource descriptions and many other actions.

For a better understanding of the embodiments, a brief review of terminology (Section 1.1) is first undertaken, then a discussion of descriptor relationships (Section 1.2), the DDF (Section 2), the serialisation syntax specification (section 3), and the DesOM API specification (section 4) used in the preferred embodiments. A more detailed description of the preferred embodiments is then given in Sections 6 to 15.

1.1 Terminology

1.1.1 Content

Content is defined to be information, regardless of the storage, coding, display, transmission, medium, or technology. Examples of content include digital and analog video (such as an MPEG-4 stream or a video tape), film, music, a book printed on paper, and a web page.

1.1.2 Resource

A resource is a particular unit of the content being described. Examples of a resource include an MPEG-1 video stream, a JPEG-2000 image, and a WAVE audio file.

1.1.3 Feature

A feature is a distinctive part or characteristic of the resource which stands for something to somebody in some respect or capacity. A feature can be derived directly (i.e., extracted) from the content (e.g., dominant colour of an image) or can be a relevant characteristic of the content. Examples of features include the name of the person who recorded the image, the colour of an image, the style of a video, the title of a movie, the author of a book, the composer of a piece of music, pitch of an audio segment, and the actors in a movie.

1.1.4 Descriptor

A descriptor associates a representation value to a feature, where the representation value can have an atomic or compound type. An atomic type is defined as one of a basic set of predetermined data types (e.g., integer, string, date, etc.). A compound type is defined to be a collection of one or more descriptors. Example descriptors having atomic types include:

Feature = Author; Representation Value (string) = "John Smith";

Feature = DateCreated; Representation Value (date) = "1998-08-08".

An example descriptor having a compound type is;

Feature = Colour; Representation Value = ColourHistogramDescriptor.

1.1.5 Description

A description is a descriptor having a compound type pertaining to a single resource.

1.1.6 Description Scheme

A description scheme is a set of descriptor definitions and their relationships (associations, equivalence, specialisations, and generalisations). The descriptor

relationships can be used to directly express the structure of the content or to create combinations of descriptors which form a richer expression of a higher-level concept. A description Scheme includes within its scope a comprehensive set of description schemes.

1.2 Descriptor Relationships

In order to express the information required for a description scheme, the DDF preferably provides means for defining a minimum set of descriptor relationships. This minimum set includes;

- Generalisation/specialisation relationships,
- Association relationships,
- Equivalence relationships,
- Spatial, temporal and conceptual relationships,
- Navigational relationships.

The generalisation/specialisation relationships specify that a particular descriptor is a more specific or more general form of another descriptor and hence can be viewed by a processing application as such. For example, a cat is a type of animal, and hence a search engine searching for occurrences of animal descriptors should also select descriptions which contain "cat" descriptors.

Association relationships are defined here to include descriptor containment and sequence and cardinality of occurrence. These relationships provide contextual information for a given descriptor and are necessary in order to provide a context in which a particular descriptor can be interpreted by an application. For example, a "Shot" descriptor which is contained within a "VideoScene" descriptor in a video description scheme would be interpreted differently from a "Shot" descriptor in another context in a sound effects description scheme.

An equivalence relationship is a form of a classification relationship where the relation is not necessarily of a generalisation/specialisation nature. Equivalence relationships are desirable between languages (i.e., inter-language) and within a language (i.e., intra-language). Typically equivalences will require the definition of synonyms (where two descriptors are equivalent) and quasi-synonyms (where two descriptors are equivalent to some specified extent). Also there is a need to define equivalence relationships between non-textual values (e.g., mean R, G and B values in an image) and a textual representative value (e.g., red, green, etc.), and vice-versa.

Spatial, temporal and conceptual relationships between descriptors in a description may also be used. These relationships support the description of neighbouring objects in an image, sequential segments in a video scene, and similar concepts in a description.

Navigation relationships between descriptors are also desirable. Usage of descriptions will often involve navigation between a component of the description and an associated spatio-temporal extent in the resource (such as a key frame in a video resource).

Considered together these relationships can to some extent provide a level of semantic interoperability between different description schemes. Further levels of semantic interoperability could also be achieved at the application level.

2. Dynamic Description Framework

2.1 Overview

The preferred DDF attempts to incorporate the benefits of declarative description of content with procedural methods for the creation and processing of descriptors. It comprises a data model, an API for the processing of descriptions, and a serialisation syntax. A DDF is able to adequately describe content using these components.

The data model provides the core semantics of the description and is based on the descriptor entity. This model has the advantage that the containment relationship is inherent in the model. This containment relationship is particularly important in the description of audiovisual resources for two reasons. First, the structure of many audiovisual resources has an inherent hierarchical structure (e.g., a video clip contains shots which contain key frames, etc.). Second, the representation values for many descriptors can be complex datatypes that can be represented in a hierarchical fashion (e.g., a histogram contains bins which contain frequencies). The data model of the preferred DDF is discussed in Section 2.2.

The preferred DDF also uses an API for the processing of descriptions. This enables non-normative applications and tools to perform further processing (e.g., transformations, presentations, etc.) on serialised descriptions. The preferred API, which is described further in Section 2.3 , is based on the DOM which is being standardised through the W3C for use with XML documents (These WC3 recommendations for XML version 1.0 and DOM version 1.0 are attached hereto as appendices L and M and were obtained on the website HTTP://www.w3.org/TR/1998/REC-xml-19980210 on the 5 January 1999 and on the

website HTTP://www.w3.org/TR/1998/REC-DOM-level1-199810001 on 28 January 1999 respectively.). The preferred API is described herein as the Description Object Model (DesOM).

The DesOM also enables the application of rule-based processing, which can:

- Extend a description by inferring the presence of additional descriptors based on the existence or absence of stored descriptors;
- Presentation of a description;
- Selecting descriptions or components of descriptions;
- Translate a stored description into another language on the basis of requirement;
- Transform a description to use a new description scheme.

This rule-based processing is described in more detail in Sections 7 to 11.

The tree-based structure of the DesOM (and for that matter, the DOM) is an appropriate representation of hierarchically structured data such as the preferred data model.

The DDF preferably uses a serialisation syntax for the purposes of storage and transport of descriptions and description schemes. This syntax expresses the data model and the processing model is able to be generated from serialised descriptions. In addition, the serialisation syntax provides a means for expressing the descriptor relationships detailed in Section 1.2. The syntax of XML Document Type Definitions (DTDs) is used to express description schemes and XML documents to serialise individual descriptions. The expression syntax of both description schemes and individual descriptions is referred to as the serialisation syntax.

The XML is used as the serialisation syntax because of its inherent ability to express the containment relationship and its increasing acceptance as a form for the transmission of structured electronic data. A description scheme can be represented using the grammar of an XML DTD in which the individual element definitions represent the definitions of the descriptors and their relationships in the description scheme. Individual descriptions can be serialised as XML documents that conform to the DTD containing the relevant description scheme. Section 2.4 describes how the preferred data model and the required descriptor relationships are expressed using the serialisation syntax.

The use of XML as the serialisation syntax enables the possibility of DDF conformant descriptions to be interpreted, in theory, at two levels. First, any serialised

description is able to be interpreted at an XML syntactical level. At this level the description could be parsed into a processing model such as the DOM and a search/filter engine with no knowledge of the DDF could interpret the description in terms of its textual content (i.e., the semantics of the DDF's data model are not used for the description's interpretation). Alternatively, the description could be parsed at a more semantic level by using the DDF data model and parsing the description in the DesOM rather than the DOM.

In practice, however, it is necessary to parse the description scheme expressed using the XML DTD syntax into an XML DTD where descriptor specialisation/generalisation relationships are validated and explicitly realised (see Section 2.4.1.1 for further details). This step is necessary because no level of subclassing or inheritance is provided for in Version 1.0 of XML. We refer to this step as DDF interpretation and the process performing the step is a DDF Interpreter. To differentiate between the DTD containing the DDF definition of a description scheme and the DTD to which the description (i.e., XML Version 1.0 document) conforms, we name the DDF DTD using an extension "ddf" rather than "dtd" as is typically used for an XML DTD.

A serialised description can then be parsed and represented using the DOM from its conformant DTD (i.e., the DTD stored using the extension "dtd") by a standard XML Processor. This processor needs no knowledge of DDF and the content of the descriptions can be accessed at a textual level. [Textual access to the description could also be achieved by simply scanning the description (XML document) or using XML Processors that are not based on the DOM (e.g., SAX;] Alternatively, a DDF Processor can parse the serialised description and represent it using the DesOM from the DTD containing the description scheme expressed using DDF (i.e., the DTD stored using the extension "ddf"). The first step of the latter process is the one of DDF interpretation.

This process of two level interpretation is depicted in Figs. 2A and 2B, which show how different semantic levels of access can be obtained from a (DDF) description serialised using the XML syntax. The DesOM differs from the DOM in that DesOM can obtain element nodes which have a richer interface than the corresponding element nodes in the DOM. In addition, the element nodes of the DesOM can have an associated DescriptorHandler (H) which provides procedures that are relevant to the element.

<u>2.2 Data Model</u>

2.2.1 Overview

The data model adopted for the preferred DDF is based on the definition of a core Descriptor object. As defined in Section 1.1.4 , a descriptor can be viewed as an "feature-representative value" pair. The representative value can be of atomic type (e.g., integer, string, date, etc.) or compound type, where a compound type is a collection of one or more descriptors. The data model is represented by the UML class diagram in Fig 3. [Note that the use of capitals in Descriptor and Description implies the objects as defined in Figure 3 rather than the general terms defined in Section 1.1.4.]

A Description object is defined as a specialisation of a Descriptor in which all the contained Descriptors pertain to a single resource. Description schemes will contain definitions of descriptors and descriptions which are specialisations of the core Descriptor and Description objects, respectively.

In the preferred data model descriptors can represent properties and relationships of their parent descriptors. For example, a Region Descriptor for a Region Adjacency Graph of an image could contain a Label Descriptor (containing a textual representative value) and a Neighbours Descriptor (containing a representative value comprising a list of references to other Region Descriptors). In this example, the Label Descriptor can be viewed as representing a property of a region and the Neighbours Descriptor as representing a spatial relationship involving the region. Descriptors representing relationships (e.g., spatial, temporal, conceptual) typically have representative values that comprise one or more references to other descriptors in the description. In Section 2.4.1.4, a standard set of descriptors are proposed to express spatial, temporal and conceptual relationships.

2.2.2 Descriptor Class

Each Descriptor has an associated id, language code and dataType enumeration. The id attribute provides each Descriptor with a unique identity. This identity can be used to reference other Descriptor objects in a description. The language code attribute specifies the language of any text in the Descriptor's representative value. The dataType enumeration provides the data type of the representative value if that value is atomic (i.e., not composed of other descriptors; see Section 2.2.3). Each Descriptor object can also be associated with a Descriptor Handler which provides procedural methods associated with the Descriptor (see Section 2.2.4).

Implementations of the preferred DDF data model can implement to various extents the descriptor relationships detailed in Section 1.2. This approach means that different implementations can utilise the properties of the particular serialisation syntaxes adopted. Section 2.4.1 describes in detail how the descriptor relationships detailed in Section 1.5 are realised using an XML serialisation syntax.

2.2.3 Atomic Descriptor Value Class

A Descriptor's representative value can be atomic or compound (i.e., composed of other Descriptor objects). If it is atomic, then the value is stored in an Atomic Descriptor Value object as a string object. The data type of this atomic value is interpreted using the dataType attribute of the parent Descriptor object. Therefore the extent to which data typing is provided depends on the dataType attribute for particular implementations of this data model. For example, refer to Section 2.4.2 for data typing implementation details using the preferred XML serialisation syntax.

The Atomic Descriptor Value could also be represented by a data attribute of the Descriptor class. The Atomic Descriptor Value is represented here as a class because of the one-to-one correspondence of this entity to a Text node in the DOM (and AtomicDescriptorValue node in the DesOM; see Section 4.1.3).

2.2.4 Descriptor Handler Class

In the preferred DDF, a Descriptor Handler is a class which provides procedural methods that apply to the Descriptor. The methods of the Descriptor Handler preferably satisfy a specified interface. The Descriptor Handler classes can provide methods for the creation of a Descriptor's representative value (or content) and the computation of the similarity between two descriptors of the same type (i.e., that use the same Descriptor definition and hence Descriptor Handler). There is no reason why this set of procedures could not be extended if required. In Fig. 3 shows some details on Descriptor Handler methods provided in the preferred implementation of the DDF.

The methods mentioned above are preferably implemented as static (class) methods that satisfy a specified interface (e.g., see Section 4.1.2). The role of the Descriptor Handler is to provide unambiguous procedures for the generation and processing of Descriptors. The ability to pass parameters to Descriptor Handler method is discussed in Section 3.1.2.1 with respect to the use of XML as a serialisation syntax.

2.2.5 Description Class

The Description has some additional attributes to those of the Descriptor. It has an associated resource which contains either the URI or ENTITY of the item of content being described. It also contains a reference to the data when that resource was last modified and an attribute that contains the URIs or ENTITIES of sets of rules that can be applied to the Description. Rule-based processing of descriptions is discussed further in Section 7.

Since a Description object is defined as a specialisation of the Descriptor object, Description objects can be treated as Descriptor objects in other descriptions(i.e., the attributes of the Description are ignored). In an alternate data model, the Description object can contain both Descriptor and Description objects. With this data model Description objects can exist in another tree of Descriptors and refer to resources other than that of the root description.

Another alternative implementation could use a data model which did not include a Descriptor object, since a Description is essentially the same as a Descriptor having a compound representative type. In this case the additional attributes of the Description (ie resource, dateResourceLastModified and ruleSets) would be treated as attributes of the Description. With this data model the resource would only need to be specified at the top of the Descriptor tree where it was relevant.

2.3 API for Processing of Descriptions (DesOM)

The inherent containment property of the core Descriptor object is represented by a tree-based processing model (i.e., parent-children data model) where each node of the tree is either a Descriptor or Atomic Descriptor Value object. [Atomic Descriptor Value objects can only exist as leaf nodes of the tree.] The processing model also contains references and navigational links between nodes in the tree. References are typically used to indicate relationships (e.g., spatial, temporal and/or conceptual) between Descriptor objects. Navigational links are used to provide browsing properties for the description and enable linking between Descriptor objects in the description and spatio-temporal extents in the resource (e.g., a particular frame in the video stream being described). A schematic depicting the description processing model is shown in Fig. 4.

For a description to conform to the preferred DDF, the root of the DesOM must be a Description object. In other words, the root must specify the resource to which the description refers. Since a Description object is just a specialisation of the Descriptor

object, any Description object can become a sub-tree of another Description object. In other words, a new Description object can be created from a set of related Description objects. This process is shown in Fig. 5.

The DesOM extends the DOM by providing the required generalisation/specialisation relationships for descriptors, data typing for atomic representative values for descriptors and reference and navigational links. The DOM provides a standard set of objects for representing XML documents, a standard model of how these objects can be combined, and a standard platform- and language-neutral interface for accessing and manipulating them. The DOM representation of an XML document is a tree structure where the content of an element is represented as child nodes of the element. The DOM specifies interfaces which can be used to manage XML documents. In other words, it can be implemented in any (or nearly all common) programming languages.

Similarly only interfaces are specified for the DesOM. These interfaces can be used to process XML documents that are DDF conformant. Just as an XML Processor must implement a DOM interface, a DDF Processor must implement a DesOM interface (see Fig. 2). As mentioned in Section 2.1, a DDF Processor must first perform an interpretation step in which the generalisation/specialisation relationships of descriptors is validated and processed in a Version 1.0 DTD form. [Invalid subclassing in the description scheme expressed using the DDF and the syntax of XML DTDs should result in a description scheme parsing error.] The DDF Processor can then either parse the description into a DOM and transform that structure into a DesOM or parse the description directly into a DesOM.

Essentially the DesOM differs from the DOM in that element and text nodes are replaced by the richer interfaces of Descriptor and Atomic Descriptor Value nodes. Interfaces for these nodes are provided in Section 4. A basic DesOM implementation could provide just that interface, however a more expansive implementation might provide some level of interpretation of the reference and navigational relationships. For example, a set of spatial, temporal and conceptual relationships could be defined for the DDF (see Section 3.1.3) and these could be interpreted at the DesOM level.

Implementations of the DesOM could optionally execute Descriptor Handler methods to create or process descriptors. For example, a DesOM implementation might

implement a Descriptor Handler's method to create the content for a Descriptor if the content did not already exist.

The DesOM provides a basis for the further processing of descriptions. The tree-structure of the DesOM makes it amenable to rule-based processing where rules consist of a pattern and an associated action. Such processing could be performed by non-normative applications and tools. These tools could be developed by implementing the DesOM interface to process DDF descriptions. Rule-based processing is discussed further in Section 7 to 11.

2.4 Serialisation Syntax

The serialisation syntax preferably used for the storage and transport of descriptions and description schemes is XML Version 1.0. The XML standard was developed as a subset of Standard Generalised Markup Language (SGML). An XML document contains one or more elements, the boundaries of which are either delimited by start and end tags, or by an empty-element tag. Each element is identified by its name, sometimes also called its "generic identifier" (GI) and may have a set of attribute specifications. Each attribute specification has a name and a value.

The preferred DDF uses a set of core elements which can be defined in an DDF Core DTD. A SGML-like DTD syntax is used to define element types and their associated attributes (as specified in the Version 1.0 XML standard). Each description can be represented by an XML document. This document (i.e., the description) refers to the DTD (i.e., the description scheme) to which the description conforms. In other words the description is of the type specified by the DTD (see Fig. 6)

The DDF Core DTD needs to provide definitions for the core elements required for the expression of the data model. The element definition that is central to the DDF is that of the Descriptor element. All descriptors can be defined as subclasses (specialisations) of this core element. For example, although a Description is defined to be a collection of descriptors pertaining to a single resource it is defined as a subclass of the Descriptor element. Other subclasses of the Descriptor element are used to provide linking functionality between the descriptors and the resources being described (see Section 3.1.4).

The data modelling requirements of the DDF are more extensive than those provided by the XML Specification version 1.0. Specifically the serialisation syntax of the DDF is able to:

- Express the required descriptor relationships (see Section 2);
- Provide data typing for the (atomic) representative value of a descriptor;

These requirements are addressed in Sections 2.4.1 and 2.4.2 with respect to using Version 1.0 of the XML standard as the serialisation syntax.

2.4.1 Expression of Descriptor Relationships

2.4.1.1 Generalisation/Specialisation Relationships

Version 1.0 of the XML specification does not provide for the specification of generalisation/specialisation relationships. In addition, subclassing and inheritance in marked up documents is not well-defined. An element type is a subclass (specialisation) of another element type, the superclass, if it is substitutable wherever the superclass element occurs and is defined to be a subclass of the superclass. It is not essential for an element to be defined as a subclass of another element. The superclass can be viewed as a generalisation of the subclass. The notion of inheritance can be viewed as a code-saving mechanism which allows one element type to get (inherit) the properties of another element type "for free".

The preferred subclassing/inheritance guidelines for single subclassing/inheritance is described below in Sections 2.4.1.1.1 to 2.4.1.1.3. Multiple inheritance can be extended from the single subclassing/inheritance.

2.4.1.1.1 Content Model Inheritance

A subclass should faithfully implement a base class's interface. Therefore, if a base class has a content model of "ANY" then a subclass can have either an "ANY" content model or a more restricted content model. This is necessary for the subclass to be substitutable for the parent class. This is a somewhat different scenario from object oriented programming (OOP) where a subclass must accept any input that its super (parent) class can. The content model of an element should be viewed as "output" not "input". If each element is considered as an object having methods to retrieve its content, then a subclass must also be able to satisfy these methods. Each element type in a content model can be viewed as having a role and the roles of a subclass's content model must match up with those of its parent class. A subclass cannot make more flexible or extend

components of the content model of its parent class, however it can implement new child elements that will be ignored when that element is treated as its parent class.

For example, if AA, BB and CC are subclasses of A, B and C, respectively and A has a content model of (B, C) then the following are all valid content models of AA; (BB, CC), (B, C), (BB, C) and (B, CC). The content models (B, C, D), (BB, CC, D) and (D, B, C) are also valid content models for AA because they match "roles" for (B), (C) and (B, C). In addition element, AA can contain child element D which will not be visible if element AA is to be treated as an instance of element A. The content models (B) and (C) are invalid because of the "role" of (B, C) in the content model of A is not matched.

It would be possible to allow the content model for a subclass to be left unspecified in which event the subclass's content model would default to be that of the superclass. Preferably,unspecified content models should not be allowed as they do not represent a valid construct using XML/SGML DTD syntax.

2.4.1.1.2 Attribute Inheritance

The same subclass and inheritance notions apply to attributes, however attributes are more intrinsically amenable to concepts of subclassing than content because they are "random access" in some sense as are methods in OOP. A subclass can declare new attributes which are essentially ignored when the subclass is treated as it parent class. However, a subclass cannot extend, or make more flexible, attributes of the parent class.

The attribute defaults are only considered when assessing whether an attribute definition has or has not extended that of its parent class. Consequently a subclass and its specified superclass should have the same attribute type, and only the attribute default can be further restricted in the subclass. Valid restrictions of attribute default definition are as in Table 1. In addition, if the superclass has a default declaration of "#FIXED" and the value of the default can be interpreted as an element name then preferably the value of the default can be further restricted to a be a subclass of that element name.

Table 1. Permitted restrictions of the attribute default declaration in a subclass.

Superclass Attribute Default Declaration	Subclass Attribute Default Declaration			
	#IMPLIED	#REQUIRED	"value"	#FIXED "value"
#IMPLIED	√	√	√	√
#REQUIRED		√	√	√

"value"			√	√
#FIXED "value"				√

2.4.1.1.3 Implementation Details

In order to implement this subclassing/inheritance model using Version 1.0 of the XML Specification and the DOM, the superclass (or superElement) for an element is specified as an attribute in the element's defined attribute list.. It is believed that this is not ideal and that subclassing information should be part of the element's definition. For example, the keyword "TYPEOF" has been suggested as a means of representing subclassing information (i.e., <!ELEMENT Cat TYPEOF Animal>).

The subclassing/inheritance implied by the use of the superElement attribute needs to be interpreted and validated against the provided guidelines for subclassing/inheritance. Failure to conform to these guidelines should result in a description scheme parsing error. Also, in order for a serialised DDF description to be a valid XML document, the description needs to conform to a valid XML DTD. Therefore the DDF description scheme that is expressed using the syntax of XML DTDs needs to be parsed to create an XML DTD in which all the inheritance aspects of the subclassing relationships are processed. This involves:

- Making explicit content models which depend on subclassing (this may involve extending content models so that they represent valid XML DTD content models in the absence of subclassing semantics);
- The addition of inherited attribute definitions to subclassed Descriptor definitions.

2.4.1.2 Equivalence Relationships

The location of described resources can be achieved by the preferred method by formulating requests directly based on a description scheme or by more unstructured queries in which the contents of a description scheme are unknown. Typically the former approach will result in a more satisfactory result because the query is specifically formulated for the form of the descriptions. However, in some cases a query might be formulated without a (complete) knowledge of a description scheme (and hence use different terms than those used in the description scheme) or in a language other than that used by particular descriptions.

As highlighted in Section 1.2 there are three types of equivalences:

- Intra-language equivalences (i.e., synonyms or quasi-synonyms);
- Inter-language equivalences (i.e., translations);
- Inferred equivalences between textual and non-textual representative values.

Known intra-language equivalences could be incorporated into a descriptor's definition using an *alias* or *sameAs* attribute for elements. However, applications and tools that provide a level of intra-language equivalence interpretation exist and therefore it was considered unnecessary to provide this functionality. Separate search/query/filter engines can ultimately provide some level of intra-language equivalence interpretation.

It is desirable to provide a means for inter-language equivalence as queries will not always be formulated in the same language as the description. Although some degree of redundancy can be tolerated in a description scheme (i.e., descriptors in different languages could be defined), it is not generally acceptable to express a description in multiple languages. The preferred method can translate a parsed description into the language of the query by processing a set of rules that is defined for the description scheme. This set of rules effectively replaces the descriptors in the DesOM with equivalent descriptors in the same language as the query. This method provides a controlled mapping between descriptors in different languages rather than allowing a mapping to be estimated by a translation ability in the search/query/filter engine.

Equivalences between non-textual and textual descriptors can be provided in a similar manner. For example, if the colour of an object in an image is stored as a (R, G, B) value then a rule could instantiate another descriptor in the DesOM that maps the particular (R, G, B) values to particular colours expressed as a text string (e.g., red, green, blue, orange, etc.).

The rules are stored as a rule set that can be specified as part of the description. The extra or translated descriptors are not serialised and are only generated when they are needed. In other words, they only exist in the DesOM and not in the XML document that represents the description. Rule sets are a way of providing a richer, more flexible, description at the time of the description being processed without increasing the overhead of storing redundant information.

2.4.1.3 Association Relationships

Association relationships specify the context in which a defined Descriptor can occur. The context includes relationships such as containment (e.g., Descriptor A must

occur within a Descriptor B), sequence (e.g., Descriptors A, B, and C must occur in that order), and cardinality (e.g., Descriptor B can occur only once in an instance of Descriptor A).

To a large extent these association relationships can be specified in an XML DTD using an element's content model. A content model is a simple grammar governing the allowed types of child elements (i.e., containment) and the order in which they are allowed to appear. Group connectors [and (comma), or (vertical bar)] are used to specify the order in which child elements can appear within the element. Occurrence indicators [one or more (+), zero or more (*), or zero or one (?)] are used to specify the cardinality or occurrence of the child elements in the element's content. Element content models are described in Section 3.2.1 of 7]. The XML content model 1.0 does not allow a specific non-zero cardinality to be defined (e.g., an image can contain 0 to 20 objects) and consequently this association property is not provided in the preferred DDF implementation.

2.4.1.4 Spatial, Temporal and Conceptual Relationships

Many descriptors will need to be able to model spatial, temporal and conceptual relationships often in addition to association relationships. For example, a Region Adjacency Graph which describes an image, comprises a graph object that contains a set of regions. In addition to being part of the graph object, each region also has a set of neighbouring regions (i.e., spatial relationships). These relationships can be described using references to the relevant descriptors in the description.

In the preferred method, these relationships are represented as Descriptors having atomic Descriptor values with IDREF or IDREFS data types. A set of core relationship descriptors is defined in the DDF Core DTD to enable DesOM implementations to realise a greater extent of semantic interpretation. Examples of the types of descriptor definition to are included are provided in Section 3.1.3.

2.4.1.5 Navigational Relationships

Many applications may require that descriptors can be explicitly linked to spatially and/or temporally localised extents in a resource. Although the resource is typically that being described, this is not always the case. The links should enable navigation from descriptors to indicated locations in a resource (e.g., from a descriptor to a spatially and temporally localised extent in a digital video stream).

The means for expressing these links has been derived from an existing approach to this problem, namely the HyTime standard, which uses location address elements, or locators. This method requires that the resource must be declared as an external entity in the description. Link elements are then declared to create contextual (having a single linkend) and independent (having more than one linkend) links between locations in the description and extents in the declared entity. Locators provide a means for addressing extents in the resource being described.

The Locator and Extent elements defined in the DDF Core DTD are much simpler than those specified in the HyTime standard as the latter provided more than was required for the DDF requirement of linking. Also, because it is difficult to envisage all the possible different forms of locators required for the different media types it was believed that description scheme designers should not be limited in the scope of their design of required locators.

2.4.2 Expression of Specific Data Types

The content model for an element can specify the order and cardinality of allowed child elements (see Section 2.4.1.3), that the element has EMPTY or no content, that the element has parsed character data (i.e., #PCDATA), or some mixture of parsed character data and child elements (i.e., ANY). [The allowed content models of elements are detailed in Section 3.2.1 of XML 1.0 WC3 recommendation]. If the content of an element is used to store the representation value of a feature (e.g., "DateCreated"), then the content model of the relevant Descriptor would need to be "#PCDATA" (or "ANY") and the content would be represented as a character string. Although this might be acceptable for a textual interpretation of the description, this form of representation does not permit more advanced queries where, for example, descriptions may be required to be selected if the "DateCreated" feature has a representation value that is later than some provided date. In other words, it is necessary to know how to parse the character content of the Descriptor (i.e., the Atomic Descriptor Value).

The serialisation syntax of the DDF provides data typing of an element's content by using a *dataType* attribute for the element. Although it would not be explicit for a Version 1.0 XML (DOM) Processor, a DDF Processor can use the data type attribute to interpret an element's content appropriately. Datatyping of element content has been considered as

part of the XML working group discussions and hence it would be preferable if the DDF could remain consistent with the XML standard.

In addition to the basic data types (e.g., integer, floating-point value, string, date, time, etc.), the *dataType* attribute should allow types such as ID, IDREF and ENTITY in order to enable Atomic Descriptor Values to represent references to other Descriptors and links to entities external to the description. The XML concept of ENTITIES is preferred to using a URI data type in that the ENTITY type allows a URI to be linked to a type of the entity (e.g., JPEG image, Java class, etc.).

2.5 Implementation Issues

An implemented DDF Processor could use publicly available software (e.g. IBM's XML parser; to parse descriptions into a DOM structure and then transform this structure into a DesOM structure. The Java language is preferably used to implement Descriptor Handler classes because of its cross-platform properties.

Actual implementations of a DDF Processor would not need to create a DOM as an intermediate step and could parse the XML document directly into a DesOM structure using the DDF description scheme. Such a processor would need to first interpret the subclassing information in the DDF description scheme (see Fig. 1).

A DDF Processor implementation could also take advantage of other core relationship descriptors (see Section 2.4.1.4) to provide a richer semantic interpretation of descriptions. Implementations could also interpret the linking elements when providing a graphical representation of descriptions and incorporate rule-engines to process rules which might be applied to the DesOM.

3. Serialisation Syntax Specification

3.1.1 Element Definitions

The preferred DDF includes the definition of a set of core elements using the XML/SGML DTD syntax. This set, is preferably stored in a core, or set of core, DTDs. Appendix A contains an example of such a DTD, *Core.ddf*. Note that we use the extension "ddf" to differentiate this document from an XML DTD which would typically have the extension "dtd". A DDF set of definitions needs to have its subclassing/inheritance properties (e.g., attribute inheritance from super elements) processed before a description can be interpreted with respect to set of DDF definitions.

The set of core elements can be used as a basis for the definition of application DTDs or description schemes. The element definitions in the *Core.ddf* effectively provide a set of "foundation" elements from which description schemes can be based.

This specification of the core element definitions for the proposed DDF is based on Version 1.0 of the XML Specification (see Appendix L). Elements that are included in the proposed *Core.ddf* are named according to the naming conventions used for Java classes (i.e., all words in the name are capitalised and concatenated).

3.1.2 Core DDF Element Definitions

3.1.2.1 Descriptor Definition

The Descriptor element is the basic element which provides the data modelling properties detailed in Section 2.2. Any element definition requiring any of these properties should be represented as a subclass of this element. The element is the markup equivalent of the object class of an object-oriented programming language.

The content model for the Descriptor element needs to allow for either parsed character data (atomic representation value) or one or more Descriptor elements (compound representative value). The content model of the Descriptor element is defined to be "ANY" so as to allow the necessary content and be a valid XML DTD syntactical construct. However in order to control content models more tightly, it is also possible to define two subclasses of the Descriptor, the Atomic Descriptor and the Compound Descriptor. The content models of these subclasses could then have the more restricted content models of #PCDATA and (Descriptor+), respectively.

In specialisations of the basic Compound Descriptor element, the "Descriptor+" would need to be interpreted by the DDF Interpreter as one or more Descriptor or subclasses of Descriptor elements. Specialisations of the Descriptor element that use this content model by default may have their content model extended to "ANY" during the DDF interpretation process (see Section 2.4.1.1.3) in order to form a valid XML DTD for a description scheme.

```
<!ENTITY % DataTypes "(Int | Float | Double | String | Date | Time | ID | IDREF |
IDREFS | ENTITY | ENTITIES)">

<!ELEMENT Descriptor (ANY)>
```

```
<!ATTLIST Descriptor
 id              ID             #IMPLIED
 xml:lang        CDATA          "en"
 dataType        %DataTypes;    "String"
 superElement    NMTOKEN        #IMPLIED
 handler         ENTITY         #IMPLIED
>
```

Attribute *id*

The value of this attribute provides a natural way to refer to a particular element (e.g., in references). It must be unique for the document.

Attribute *xml:lang*

The attribute *xml:lang* is included in Version 1.0 of the XML Specification. It specifies the natural language or formal language in which the content (of the element) is written. The default language used by the Descriptor element is English. If a description scheme was defined in French, for example, then one approach would be to define a FrenchDescriptor in which the value of *xml:lang* was FIXED to "fr", and then derive all application descriptors from the FrenchDescriptor element

Attribute *dataType*

Preferably, the definitions of many descriptors require some control over the data type of an element's character data content.

The allowed data types for character data content are specified by the (XML) internal parameter entity, DataTypes (see above). The *dataType* attribute is only utilised if the content model for the Descriptor contains #PCDATA and the provided content for the Descriptor contained character data. In other words, if the content of a Descriptor is specified to contain child elements (i.e., a compound representative value) then the *dataType* attribute is not used. In an alternative implementation, the allowable data types could include a "Compound" type which would make the use of a compound descriptor more explicit in its definition.

Character data content of a Descriptor is represented by a DDF Processor using a AtomicDescriptorValue node (see Section 4.1.3 for the interface specification) rather than a Text node as used by a DOM Processor.

The default value of the attribute *dataType* is "String". This means that the *dataType* attribute does not need to be included in a Descriptor element's definition if the content of the element is to be treated as a string. Preferably, the DDF Processor dates and times are based on the profile of ISO 8601. The types, ENTITY/ENTITIES/ID/IDREF should be parsed as defined for the XML Version 1 standard [see Appendix L].

Although the data type of the Descriptor element's character data content cannot be directly used by XML version 1.0 and DOM version 1.0 specifications, it might in some way assist textual access to the description. Also placing the data type of the character data content in an attribute is consistent with many current proposals for data typing in XML.

Some Descriptors will require their representative value to be limited to a list of possible values (i.e., an enumeration). In these cases, it is preferable to construct Descriptor elements (having an EMPTY content model) for each of the enumerated values and then specify the enumeration in the content model for the parent Descriptor. An alternate approach is to include an enumeration data type and use a #PCDATA content model.

Attribute *superElement*

The value of this attribute is an element name which is the parent (or super) element of the Descriptor element. The parent element's definition must be available. Subclassing is implemented as described in Section 2.4.1.1.

The information in this attribute is used by the DDF interpretation process (see Fig. 1) to validate the defined subclassing and to process the inheritance of attributes. When accessed at the DOM level, this attribute provides only descriptive information about the immediate generalisation of the element. When processed at a DesOM level the subclassing relationship(s) for the element are represented as a node list or inheritance tree (see Section 4.1.1).

Attribute *handler*

The value of this attribute specifies an external entity for a Descriptor Handler to be used to provide methods for the Descriptor element. The Descriptor Handler is a class

which contains methods that conform to a specified Descriptor Handler interface (see Section 4.1.2).

The Descriptor Handler is specified using an ENTITY which can be defined in the description scheme (preferably before the elements of the scheme are defined). The ENTITY declaration can use a NOTATION to declare the type of the external entity and a helper application required to process the ENTITY. In the example below, a NOTATION is declared for a JavaClass type and this type is linked to the "Java" helper application (i.e., Java virtual machine). An individual Java class in then declared using an ENTITY declaration which uses the JavaClass NOTATION.

```
<!NOTATION JavaClass SYSTEM "Java">
<!ENTITY MyDescHandler SYSTEM "MyDescHandler.class" NDATA JavaClass>
```

Preferably, it is assumed that the methods provided by the Descriptor Handler do not require any parameters that are not available from the DesOM (e.g., resource from a Description element). If methods of a Descriptor Handler require parameters to be set from individual descriptions, then attributes of a specialisation of the Descriptor element can be used to hold the parameter values. A Descriptor Handler could then have a method to set the parameters from the attribute values in the DesOM.

3.1.2.2 Description Definition

A Description element is defined as a subclass of the Descriptor element. It represents the root node of an instance of the DesOM and should be the root element of a serialised description (i.e., an XML document).

The content model for the element is defined as one or more Descriptors. This is a restriction of the content model of the Descriptor element. As with the Descriptor element, definitions of specialisations of this element need to be interpreted by the DDF Interpreter as one or more Descriptor or Descriptor subclass elements.

```
<!ELEMENT Description (Descriptor+)>
<!ATTLIST Description
    superElement              NMTOKEN     #FIXED "Descriptor"
    resource                  ENTITY      #REQUIRED
    dateResourceLastModified  CDATA       #IMPLIED
    ruleSets                  ENTITIES    #IMPLIED
```

>

Attribute *superElement*

Although the attribute *superElement* is inherited from the Descriptor element's definition, it is redefined here to declare that the Description element is a subclass of the Descriptor element. The default superElement is declared as #FIXED so that instances of the Description element cannot redefine the superElement value. Note, that a specialisation of the Description element can further restrict this default attribute value by specifying an element name that is a subclass of the Descriptor element (see 2.4.1.1.2).

Attribute *resource*

This value of this attribute should contain an entity which references the resource being described by this description. The resource must have been declared as an entity in the description before the Description can be declared. The resource type can be obtained by using a NOTATION, defined in either the description scheme or in the *Core.ddf*, to describe the type of entity:

e.g., <!NOTATION MPEG-2 SYSTEM "MPEG-2Player">.

The NOTATION can then be used by an external ENTITY declaration in the DOCTYPE declaration of the description:

e.g., <!ENTITY MyVideo SYSTEM "MyVideo.mpg" NDATA MPEG-2>.

Note, that this method of referencing the resource being described not only identifies it as an MPEG-2 resource but also provides the name of a processor (helper application) for the resource type.

Attribute *dateResourceLastModified*

The value of this attribute is a string representation of the date that the resource was last modified. At any stage a process can check to see if this date has changed (by string comparison), and update the description if necessary.

Attribute *ruleSets*

The value of this attribute contains one or more external ENTITIES. Each ENTITY refers to an XML document that contains a set of rules that can be applied to the description (see Section 7).

<u>3.1.3 Descriptors Representing Spatial, Temporal and Conceptual Relationships</u>

A set of Descriptor elements have been included to provide spatial, temporal and conceptual relationships between descriptors. These elements are preferably a part of the core DDF elements rather than specified in individual application description schemes in order to improve the semantic interpretation of description. These relationship Descriptor elements can have either atomic or compound representation values. The element set below is included more by way of example rather than attempting to demonstrate a complete list of the types of relationships that need to be modelled.

```
<!ELEMENT ParallelSequence (Descriptor+)>
<!ATTLIST ParallelSequence
    superElement        NMTOKEN         #FIXED "Descriptor"
>


<!ELEMENT SerialSequence (Descriptor+)>
<!ATTLIST SerialSequence
    superElement        NMTOKEN         #FIXED "Descriptor"
>
<!ELEMENT Neighbours (#PCDATA)>
<!ATTLIST Neighbours
    superElement        NMTOKEN         #FIXED "Descriptor"
    dataType            %DataTypes;     #FIXED "IDREFS"
>
<!ELEMENT Before (#PCDATA)>
<!ATTLIST Before
    superElement        NMTOKEN         #FIXED "Descriptor"
    dataType            %DataTypes;     #FIXED "IDREFS"
>
<!ELEMENT After (#PCDATA)>
<!ATTLIST After
    superElement        NMTOKEN         #FIXED "Descriptor"
    dataType            %DataTypes;     #FIXED "IDREFS"
>
```

```
<!ELEMENT InFrontOf (#PCDATA)>
<!ATTLIST InFrontOf
    superElement        NMTOKEN         #FIXED "Descriptor"
    dataType            %DataTypes;     #FIXED "IDREFS"
>
<!ELEMENT Behind (#PCDATA)>
<!ATTLIST Behind
    superElement        NMTOKEN         #FIXED "Descriptor"
    dataType            %DataTypes;     #FIXED "IDREFS"
>
```

3.1.4 Elements Representing Navigational Relationships

The preferred DDF also includes some core elements that enable the linking of descriptions to spatio-temporal extents of the content being described. A spatio-temporal extent is defined to be a section of the content that is spatially and/or temporally localised. For example, a spatio-temporal extent of a digital video signal might be represented as a rectangular region that extends for a number of frames. It is proposed to use an adaptation of the core linking elements from the HyTime standard. In particular, a contextual link, CLink, is defined to represent the common cross-reference or navigational link. Like in HyTime, a CLink connects the location in the description where the link occurs to another location. In other words, a CLink has a single linkend attribute. An independent link, or ILink, is also defined for applications that require links connecting more than two locations or stored separately from the link's location in the description. These elements are defined as subclasses of the basic Descriptor element so that they are interpreted by the DDF and represented as nodes in the DesOM. Since these elements do not require any of the data modelling properties described in Section 2.2, there may be a case for allowing elements, such as the set defined below, to not be based on the Descriptor element but still interpreted by a DDF processor.

The definitions of these linking elements are included in *Core.ddf*. Note it might be preferable to include the definition of the core spatio-temporal linking elements in a separate (ddf) DTD just as in HyTime core elements are defined in separate DTD modules and can be included as required in an application DTD.

```
<!ELEMENT CLink (#PCDATA)>
<!ATTLIST CLink
 superElement        NMTOKEN        #FIXED "Descriptor"
 dataType            %DataTypes;    #FIXED "IDREF"
>
<!ELEMENT ILink (#PCDATA)>
<!ATTLIST ILink
 superElement        NMTOKEN        #FIXED "Descriptor"
 dataType            %DataTypes;    #FIXED "IDREFS"
>
```

The definitions of these linking elements have been modified from their HyTime equivalents to include the linkend(s) to be specified as the atomic representation value of the link descriptor elements.

The core Locator element simply provides an address for the location of one or more Extent elements within a particular resource. The value of the *resource* attribute identifies the resource using an ENTITY that has been previously declared in the description. This requires that the *Core.ddf* also includes a sufficiently rich set of NOTATIONS that include the types of resources that are going to be referenced by entities (e.g., JPEG, TIFF, MPEG-1, MPEG-2, etc.). An instance of a Locator must contain one or more instances of an Extent. It is desirable to specify the resource even if it is the same resource specified for the description.

Several subclasses of Extent elements are defined in the *Core.ddf*. The definitions of these elements are included below. These element definitions provide an example of the types of Locator and Extent elements that could be required.

```
<!ELEMENT Locator (Extent+)>
<!ATTLIST Locator
 superElement        NMTOKEN        #FIXED "Descriptor
 resource            ENTITY         #REQUIRED
>
```

```
<!ELEMENT Extent (Descriptor+)>
<!ATTLIST Extent
superElement            NMTOKEN         #FIXED "Descriptor"
>
<!ELEMENT ImageExtent (Descriptor+)>
<!ATTLIST ImageExtent
 superElement           NMTOKEN         #FIXED "Extent"
>
<!ELEMENT RectImageExtent (RectImageExtentX0, RectImageExtentY0,
        RectImageExtentHeight, RectImageExtentWidth)>
<!ATTLIST RectImageExtent
 superElement           NMTOKEN         #FIXED "ImageExtent"
>
<!ELEMENT RectImageExtentX0 (#PCDATA)>
<!ATTLIST RectImageExtentX0
 superElement           NMTOKEN         #FIXED "Descriptor"
 dataType               %DataTypes;     #FIXED "Int"
>
<!ELEMENT RectImageExtentY0 (#PCDATA)>
<!ATTLIST RectImageExtentY0
 superElement           NMTOKEN         #FIXED "Descriptor"
 dataType               %DataTypes;     #FIXED "Int"
>
<!ELEMENT RectImageExtentHeight (#PCDATA)>
<!ATTLIST RectImageExtentHeight
 superElement           NMTOKEN         #FIXED "Descriptor"
 dataType               %DataTypes;     #FIXED "Int"
>
<!ELEMENT RectImageExtentWidth (#PCDATA)>
<!ATTLIST RectImageExtentWidth
 superElement           NMTOKEN         #FIXED "Descriptor"
 dataType               %DataTypes;     #FIXED "Int"
```

```
>

<!ELEMENT VideoExtent (VideoExtentStart, VideoExtentEnd, ImageExtent?)>
<!ATTLIST VideoExtent
 superElement          NMTOKEN          #FIXED "Extent"
>
<!ELEMENT VideoExtentStart (#PCDATA)>
<!ATTLIST VideoExtentStart
 superElement          NMTOKEN          #FIXED "Descriptor"
 dataType              %DataTypes;      #FIXED "Int"
>
<!ELEMENT VideoExtentEnd (#PCDATA)>
<!ATTLIST VideoExtentEnd
 superElement          NMTOKEN          #FIXED "Descriptor"
 dataType              %DataTypes;      #FIXED "Int"
>
```

4. DesOM API Specification

The DesOM interface extends the existing DOM interface specification. The required extensions are detailed in this Section using the Object Management Group (OMG) Interface Definition Language (IDL). The specified interface represents a minimal interface for the DesOM.

4.1.1 Interface Descriptor

The Descriptor node object in the DesOM is a subclass of the DOM Element node object (see p.32 in Appendix M). Like the Element node object, the Descriptor node object represents both the Descriptor element, as well as any contained elements (see [Appendix M] for further details on the IDL specification of the Element object).

IDL Definition

```
interface Descriptor : Element {
 void                  setHandler(in DescriptorHandler handler);
 DescriptorHandler     getHandler();
 NodeIterator          getSuperElements();
```

```
};
```

Method *setHandler()*

Set the DescriptorHandler for this Descriptor node object. This handler can be instantiated on the basis of the handler ENTITY that is specified as the value of the *handler* attribute for the Descriptor element.

Parameters

handler	The DescriptorHandler to be assigned to this Descriptor node.
Returns	void
Exceptions	This method throws no exceptions.

Method *getHandler()*

Returns the DescriptorHandler for this Descriptor node object.

Parameters	None
Returns	DescriptorHandler for the Descriptor node object.
Exceptions	This method throws no exceptions.

Method *getSuperElements()*

Returns a list of Descriptor generalisations or superElements for the Descriptor node object.

Parameters	None
Returns	NodeIterator
Exceptions	This method throws no exceptions.

4.1.2 Interface DescriptorHandler

The DescriptorHandler object provides methods for a class of Descriptor nodes. A DescriptorHandler can provide methods for more than one type of Descriptor. For example, a collection of Descriptors might use the same similarity metric.

The methods of a DescriptorHandler are generally implemented as class (static) methods.

IDL Definition

```
interface DescriptorHandler {
```

```
boolean     canCreateDescriptorContent();
void        createDescriptorContent(Descriptor descriptor, Entity resource);
void        removeDescriptorContent(Descriptor descriptor);
double      getSimilarity(Descriptor descriptor1, Descriptor descriptor2);
};
```

Method *canCreateDescriptorContent()*

Returns true if the DescriptorHandler contains an implemented method that can create the content for a descriptor..

Returns True if a method has been implemented else returns false.

Method *createDescriptorContent()*

Generates the content (i.e., child nodes) of the specified Descriptor node object using the specified resource.

Parameters

descriptor	The Descriptor node object for which the content (i.e., child nodes) is to be created from the resource.
resource	The resource, represented as an entity, from which the content is to be derived.
Returns	void
Exceptions	This method throws a ResourceNotFoundException if the resource could not be found, or a IllegalResourceException if the resource is not compatible with the method.

Method *removeDescriptorContent()*

Removes the content (i.e., child nodes) of the specified Descriptor node. This method might be invoked to reduce the complexity of a description for storage and would typically only be invoked if the DescriptorHandler was capable of re-creating the specified descriptor's content.

Parameters

descriptor	The Descriptor node object for which the content (i.e., child nodes) is to be removed.

Returns void

Method *getSimilarity()*

Returns a similarity metric in the range of [0, 1.0] which provides a measure of the similarity between the two specified Descriptor node objects.

Parameters

descriptor1	The first of the two Descriptor node objects to be compared.
descriptor2	The second of the two Descriptor node objects to be compared.
Returns	double
Exceptions	This method throws an UnmatchedDescriptorException if the two Descriptor node objects are of incompatible types.

4.1.3 Interface AtomicDescriptorValue

The AtomicDescriptorNode object is a subclass of the Text (node) object that is specified as part of the DOM [The Text object contains the non-markup content of an Element]. It provides additional methods to the Text object which interpret the string data content of the Text object as other data types (i.e., it is effectively a typed text node). The data types available are as specified for the *dataType* attribute of the Descriptor element (see Section 3.1.2.1). It is assumed in this specification that the XML data types (i.e., Ids, IDREFs, ENTITY, ENTITIES) would be interpreted from the string value of the AtomicDescriptorValue node.

Dates and times are represented using the date an time formats specified by the profile of ISO 8601. Implementations of the AtomicDescriptorValue object can provide further methods that provide extra date functions (e.g., getDataAsDateYear(), getDataAsDateMonth(), etc.).

IDL Definition

```
interface AtomicDescriptorValue : Text {
 int            getDataAsInt();
 float     getDataAsFloat();
 double         getDataAsDouble();
 Date      getDataAsDate();
 Time      getDataAsTime();
```

```
};
```

Method *getDataAsInt()*

Returns the value of the Text node as an integer.

Parameters	None
Returns	Integer
Exceptions	This method throws a DDFDataFormatException if the character string could not be parsed as an integer.

Method *getDataAsFloat()*

Returns the value of the Text node as a float value.

Parameters	None
Returns	Float
Exceptions	This method throws a DDFDataFormatException if the character string could not be parsed as a float value.

Method *getDataAsDouble()*

Returns the value of the Text node as a double value

Parameters	None
Returns	Double
Exceptions	This method throws a DDFDataFormatException if the character string could not be parsed as a double value.

Method *getDataAsDate()*

Returns the value of the Text node as an ISO 8601 date.

Parameters	None
Returns	ISO 8601 date
Exceptions	This method throws a DDFDataFormatException if the character string could not be parsed as an ISO 8601 date.

Method *getDataAsTime()*

Returns the value of the Text node as an ISO 8601 time.

Parameters	None

Returns ISO 8601 time

Exceptions This method throws a DDFDataFormatException if the character string could not be parsed as an ISO 8601 time.

5. Example of a Description Scheme

An example of a description scheme expressed in DDF is contained in Appendix B. The description scheme aims to provide a description for digital video footage of an Australian Football League (AFL) game. This description scheme makes use of some core element definitions that are contained in Appendix A. The Core.ddf is declared as an internal parameter entity B1 and then included in the description scheme using the % operator (see B2). The indicated lines B1 and B2 of the description scheme result in all the element definitions included in Appendix A being available to the example description scheme.

In the definition of the descriptor AFLGameDescription B3 a descriptor handler B4 is specified. In the preferred embodiment this descriptor handler is implemented as a Java class (AFLGameGen.class in the example contained in Appendix B) having a predetermined procedural method which automatically generates the (description) content for the AFLGameDescription descriptor by analysing the digital video signal containing the footage of the game being described.

It should be noted that although the AFLGameDescription element is defined as a specialisation of a Description element, a Description element is just a specialisation of a Descriptor element, and so the AFLGameDescription can also be treated as a Descriptor.

An example description generated from the description scheme contained in Appendix B is shown in Appendix C. This example description would typically have been initially generated by the descriptor handler for the AFLGameDescription descriptor, however manual creation is also possible if an annotator so desires. The procedural method to generate the content for the descriptor AFLGameDescription would typically analyse the digital video resource signal containing the footage of the game to be described, identify the start and end of the four quarters of play, and within each quarter track and, if possible, identify individual tracked players. The tracking could be achieved using motion analysis of the digital video resource with player identification being

achieved by attempting to recognise a player's number from his/her jersey. It is not an object of this invention to specify a method for generating the content of the description.

Clearly it is unlikely that all the information required for the description, as specified by the description scheme, could be automatically generated from an analysis of the digital video resource signal. Where information is not available (e.g., date and location of the game), the content generation method can either generate empty descriptors or simply omit the descriptors from the description. At a later date an annotator can add this information manually if it is required. Similarly, it might be too difficult for an automatic analysis to classify the action of each tracked player. For example, it might be difficult to automatically analyse whether the player was involved in a mark, a kick or a tackle. This information could also be provided at a later date. In fact, an annotator could use a Digital Video Browser System, as described in Section 11, to browse the digital video resource and annotate as required. On completion of annotation the Digital Video Browser System could also be used to select to play all those sections of the digital video resource in which a particular player was involved, or all those sections in which a mark occurred. In other words, the Digital Video Browser System could be used to complete any annotation tasks and browse the described digital video resource.

Another example of a method to create the content for a descriptor, is one where the resource to be described has already been described using another description scheme. For example, a digital video camera might generate a description (using, for example, a Video Capture Description Scheme) for a digital video resource as it is being captured. The automatically generated description might contain information such as exposure, focus, eye-gaze location, shot boundaries, etc. It might be desirable to maintain some, if not all, of the information automatically recorded using the source description scheme, however it might be preferable to describe the digital video resource using another more generally accepted description scheme, in this case the destination description scheme. In this case the descriptor handler(s) in the destination description scheme could provide a mapping of descriptors from the source to destination descriptions. This mapping would typically be provided in the content creation method of descriptor handler for the Description element of the destination description scheme. This transformation from one description scheme to another could also be achieved by applying rules to the DesOM (see Section 7).

6. Methods of Applying Procedures

6.1 Method of Applying Procedures to Electronically-Accessible Resources

Turning now to Fig. 7A, there is a shown a method of applying procedures to an electronically accessible resource. The method commences at step 700A and continues at step 702A where a description scheme is applied to a resource. In the next step 704A, a processor identifies the one or more DescriptorHandlers in the description scheme and afterwards the method continues to step 706A. In step 706A, the processor identifies the procedures corresponding to the previously identified DescriptorHandlers. These procedures are in the form of procedural code contained in the DescriptorHandlers. In the next step 708A the procedures are applied to the resource. The method then outputs at step 710A the results of the application of the procedures. The method terminates at step 712A. Preferably, these procedures result in the automatic generation of a description of the resource which may serialised as a XML document. However other procedures or processes may be envisaged. Further this resultant description is preferably interpretable by both humans and machines. In another example the DescriptorHandler can provide a method for computing the similarity between two instances of a descriptor of the resource.

6.2. Methods of Applying Procedures to a Description

Turning now to Fig. 7B, there is shown a flow diagram of a preferred method of processing a description of a resource. The method commences at step 700B and continues at step 702B where a description is parsed by a DDF processor. In the next step 704B, the DDF processor identifies within the associated description scheme one or more DescriptorHandlers. In the next step 706B of the method, the DFF processor identifies the one or more procedures associated with the previously identified DescriptorHandlers. These procedures are in the form of procedural code contained in the DescriptorHandlers. In the next step 708B the procedures are applied to the DesOM corresponding to the description. The method then outputs at step 710B the results of the application of the procedures. The method terminates at step 712B. The method envisages many different types of procedures that can used in the preferred method. In one embodiment, the preferred method computes the similarity between two same descriptors types. In this embodiment, the descriptions are parsed by the DDF processor and a common descriptor definition is identified by the processor. The DDF processor then identifies within the description scheme containing the common descriptor definition an associated

DescriptorHandler which contains procedural code for computing similarity between two descriptors. The method then applies the procedural code to the DesOMs associated with the descriptions and determines how similar the descriptors and hence the similarity of the two resources. The method then outputs the results of the similarity computation. This embodiment has particular application in searching/querying descriptions of resources.

7. Rule-based Processing using the DesOM

The internal memory structure of a description (i.e., the DesOM) provides a convenient structure on which to perform further processing of a description (or indeed the relevant description scheme). This further processing can be achieved by locating patterns of nodes in the DesOM and performing specified actions in response to the located patterns. Each pattern-action association can be represented by a rule and a set of related rules can be collected into a rule set.

Rules can be used to used to automatically create further descriptors based on existing descriptors (see Section *8. Method of Extending Descriptions of Resources*), to provide presentation properties for descriptions and description schemes (see Section *9. Method of Presenting Descriptions of Resources*), and to represent queries (see Section *10. Method of Selecting Resource Descriptions*). Rules can also be used to translate a description to the language of the query (see Section *11. Method of Translating a Description of a Resource*). The preferred embodiment of the Digital Video Browser System described in Section 12 uses a method for formulating rules common for each of these functions. This method is described below.

Each rule consists of a pattern (of nodes in the DesOM) and an associated one or more actions. For each of the different functions (inference, equivalence, presentation and selection), a different set of actions is often applicable. However each of these functions can be enabled using a common rule grammar which will be described in this section. The rule grammar can be defined in an XML DTD. The rules for the different functions can simply use the common rule grammar (this is the case for the preferred embodiment of the Digital Video Browser System), or alternatively the allowable actions can be controlled by defining different DTDs for each of the different functions (e.g., an InferenceRules.dtd, a PresentationRules.dtd, etc.).

Rules can be represented as, or in a manner similar to, Extensible Style Language (XSL) rules. In the preferred embodiment of the Digital Video Browser System (see Section 12), we have used the following basic rule grammar.

Rules.dtd

```
<!ELEMENT Rule (Action+)>
<!ATTLIST Rule
        target      (Element | ElementDefn)     "Element"
        pattern     CDATA                       #REQUIRED
>
<!ELEMENT Action (   AddAttribute | RemoveAttribute
                   | AddElement | RemoveElement
                   | AddAttributeDef | RemoveAttributeDef
                   | Select)>
<!ELEMENT AddAttribute (EMPTY)>
<!ATTLIST AddAttribute
        attName     CDATA                       #REQUIRED
        attValue    CDATA                       #REQUIRED
>
<!ELEMENT RemoveAttribute (EMPTY)>
<!ATTLIST RemoveAttribute
        attName     CDATA                       #REQUIRED
>
<!ELEMENT AddElement(#PCDATA)>
<!ATTLIST AddElement
        position    (SiblingBefore | SiblingAfter | AsFirstChild | AsLastChild )
                                                #REQUIRED
>
<!ELEMENT RemoveElement (EMPTY)>
<!ELEMENT AddAttributeDef (EMPTY)>
<!ATTLIST AddAttributeDef
        attName     CDATA                       #REQUIRED
        attType     CDATA                       #REQUIRED
```

```
        attDefault  CDATA                       #REQUIRED
>
<!ELEMENT RemoveAttributeDef (EMPTY)>
<!ATTLIST RemoveAttributeDef
        attName     CDATA                       #REQUIRED
>
<!ELEMENT Select (EMPTY)>
<!ATTLIST Select
        attName         CDATA           "selected"
        attValue        CDATA           "YES"
        selectAncestors (YES | NO)          "YES"
>
```

Each Rule element has a target attribute that has a default value of "Element" and a character string pattern attribute. The target attribute refers to the target of the defined Rule. Typically inference, equivalence and search rules are targeted at elements because the action of the rule results in either a new descriptor in the description or the selection of a descriptor for a query. Presentation rules, however are typically targeted at element definitions as their associated actions specify how a particular descriptor type is to be presented in an application. A set of rules can be serialised in an XML document. This is typically the case with inference, equivalence and presentation rules, but may not be required for selection rules which may often be processed on a single rule basis.

The role of the pattern character data string is to identify the particular elements (or element definitions) to which the action is applied. This character string can identify more than one element and can include element ancestry and attribute qualifiers. In the preferred embodiment the pattern string is parsed according to the following Extended Backus-Naur Form (EBNF) notation.

```
Pattern             ::=ElementPatterns(ConnectorOpElementPatterns)*
ElementPatterns     ::=ElementPattern (AncestryOp ElementPattern)*
ConnectorOp         ::=   '|' | '&'
AncestryOp          ::=   '/' | '//'
```

Each pattern can consist of one or more alternative patterns (i.e., ‘|’ represents an alternative) or must satisfy more than one ElementPattern (i.e., ‘&’ connector operation). Element ancestry is represented within a pattern by using the parent operator ‘/’. Two patterns separated by a parent operator match an element if the right hand side matches the element and the left hand side matches the parent of the element. For example, the following Shot elements that have a Scene element as a parent and a VideoClipDescription element as a grandparent match the following Rule’s pattern:

```
<Rule pattern = “VideoClipDescription/Scene/Shot”>
    <Action> etc…</Action>
</Rule>
```

Two patterns separated by the ancestry operator ‘//’ match an element if the right-hand side that matches the element has at least one ancestor that the left-hand side matches. So, for example, any Shot elements that have a VideoClipDescription as an ancestor element will match the following Rule’s pattern:

```
<Rule pattern = “VideoClipDescription//Shot”>
    <Action> etc…</Action>
</Rule>
```

```
ElementPattern            ::=  ElementTypePattern ElementQualification
ElementTypePattern        ::=  OneElementTypePattern | ‘*’
OneElementTypePattern     ::=  ElementTypeName
ElementQualification      ::=  ‘[‘Qualifiers?’]’
Qualifiers                ::=  Qualifier (‘,’Qualifier)*
Qualifier                 ::=  ChildQualifier
                               | AttributeQualifier
                               | PositionalQualifier
AttributeQualifier        ::=  AttributePattern (‘=’AttributeValue)?
AttributePattern          ::=  ‘attribute’ ‘(‘AttributeName’)’
AttributeValue            ::=  ‘”’ [^”]* ‘”’ | “’” [^’]* “’”
PositionalQualifier       ::=  Position ‘(‘ ‘)’
Position                  ::=  ‘FirstOfType’ | ‘NotFirstOfType’
                               | ‘FirstOfAny’ | ‘NotFirstOfmany’
```

| 'LastOfType' | 'NotLastOfType'
| 'LastOfAny' | 'NotLastOfAny'
| 'OnlyOfType' | 'NotOnlyOfType'
| 'OnlyOfAny' | 'NotOnlyOfAny'

An element within the pattern hierarchy may have qualifiers applied to it, which further constrain which elements match the term. These qualifiers may constrain the element to have certain attributes or sub-elements or may constrain its position with respect to its siblings. The qualifiers are specified in square brackets following the ElementTypeName (which is it tag name defined in the DTD). A pattern matches only if all of the qualifiers are satisfied.

For example, any Shot elements that have a child element KeyFrame will match the following Rule's pattern:

```
<Rule pattern = "Shot[KeyFrame]">
      <Action> etc...</Action>
</Rule>
```

Attributes on the target element or any of its ancestor elements can also be used to determine whether a particular rule applies to an element. An attribute qualifier can constrain an element to have either a specific attribute with a specific value, or to have a specific attribute with any value. For example, the following pattern matches a Bin descriptor which has as its parent a Histogram descriptor which has an attribute noBins with a value of '100':

```
<Rule pattern = "Histogram[attribute(noBins)='100']/Bin">
      <Action> etc...</Action>
</Rule>
```

Positional qualifiers can also be used to further constrain the pattern to match on the element's position or uniqueness amongst its siblings. For example, the following example matches Object descriptors which are the only Objects in a KeyFrame descriptor:

```
<Rule pattern = "KeyFrame/Object[OnlyOfType()]">
      <Action> etc...</Action>
</Rule>
```

The above description of the matching method permits pattern matching only on elements (which are typically descriptors in the DesOM) or element definitions. Clearly there are many possible embodiments for defining the syntax of the node pattern matching without departing from the spirit and scope of the invention.

Each Rule can have one or more associated Action elements. In the preferred embodiment of the Digital Video Browser System the allowable Action elements for rules has been limited to the addition and removal of elements and attributes from elements (i.e., descriptors) in descriptions and the addition and removal of attribute definitions from element definitions in a description scheme. The actions involving individual descriptions are generally used by inference, equivalence and selection rules (see Sections 8 and 10) and the actions involving description schemes are generally used by presentation rules (see Section 9).

The attributes of the Action elements, AddAttribute and RemoveAttribute, specify the attribute to be added or removed from a target element (i.e., an element that has matched the specified pattern in the rule). The content of the AddElement action contains the element to be added to the DesOM as a relation of a target element. The position attribute of the AddElement element specifies where the new element should be added with respect to the target element. This position attribute can indicate that the new element is to be added as a sibling node before the target element (SiblingBefore), as a sibling node after the target element (SiblingAfter), as the first child of the target element (AsFirstChild), or as the last child of the target element (AsLastChild). Clearly, since the element to be added to the DesOM is represented as parsed character data (#PCDATA), an element hierarchy can also be added to the DesOM. The RemoveElement action will simply remove a target element. Any child elements of the target element will also be removed.

The AddAttributeDef and RemoveAttributeDef actions are only valid if the target for the rule is an element definition. These actions are typically used by presentation rules (see Section 9). The AddAttributeDef action uses the attName, attType and attDefault attributes to specify the required information for the attribute definition to be added to an element definition. The RemoveAttributeDef action will simply remove the attribute definition that is identified by the value of the attName attribute of the action. Attribute

definitions can be replaced by including both an AddAttributeDef and a RemoveAttributeDef action in a particular rule.

The Select action is typically only used by selection rules and is described in detail Section 10. Rules can also be used to transform a description. These rules are used to generate a second description conforming with a second description scheme.

8. Method of Extending Descriptions of Resources

Given a description scheme, it is possible that further descriptors can be automatically created by inference or a known equivalence in a description based on the existence or otherwise of a particular set of descriptors. For example, if a descriptor for a digitally captured image representing light exposure levels indicated outdoor lighting levels, then an additional descriptor could be automatically created to classify the image as an "Outdoor Scene". Since the latter classification can be inferred from the recorded light exposure levels there is no advantage in storing the classification because it can always be re-generated while the inference rule exists. Rules can also be used to generate textual descriptors based on non-textual descriptors or vice versa. For example, the colour of an object might be stored in a description as a (R, G, B) value. A rule could be formulated which maps each (R, G, B) value to one of a possible number of colours represented in a text string (e.g., red, green, purple, etc.). The additional descriptors generated by inference or equivalence rules can result in a richer description that can be exploited by applications (e.g., search engines, filter agents, etc.).

A set of rules that is applicable for a given description scheme can be serialised (stored) in an XML document. In the preferred embodiment of the Digital Video Browser System, a reference to such an XML document is stored in the value of the ruleSets attribute of the Description element for the description scheme (see Section *3.1.2.2 Description Definition*). It is possible to associate more than one rule set with a description scheme. In the preferred embodiment of the Digital Video Browser System, if more than one rule set is specified then it is assumed that both rule sets can be applied (i.e., the individual rule sets do not contain unresolvable rules). In other words, the individual rule sets are simply combined and treated as a single rule set, in which the order of rules to be processed is provided by the order of the listing of the individual rule sets and the order of the individual rules within each given rule set. Inference and equivalence rule sets can also be stored with an application without departing from the

essence of the invention, however in this event the value of the rules is limited to the particular application.

In the preferred embodiment the Action elements typically used are the addition and removal of attributes and elements from the DesOM. Replacement can be achieved by using a removal followed by an addition Action element.

A set of inference rules is preferably invoked whenever a description is first processed into the DesOM. The rules are iteratively processed until no further changes can be made to the DesOM as some rules may depend on the actions of other rules. The rule set may need to be (iteratively) reapplied whenever the description is updated (e.g., a manual annotation in an application utilising the description). In the event that an application has permitted changes to be made to the description, then before serialising the altered description each change needs to be considered in light of the inference rules in order to ascertain whether the descriptor can be inferred from a knowledge of the other descriptors in the description. If a descriptor can be inferred then it is excluded from the serialised description.

The preferred method preferably associates a set of inference and/or equivalence rules to a description scheme. This set of rules can be implemented according to the abovementioned description and results in a richer description structure without any additional storage or transport overhead which would result if the extra (inferred or equivalent) descriptors were included as part of the individual descriptions. Being able to represent this inferred or equivalent information as a set of rules that can be invoked when required represents a significant saving in storage and transport cost if a large digital library were to be described. In other words it can eliminate the storage and processing costs of redundant information.

An important aspect of the preferred method is that unlike existing stylesheet languages such as XSL, the inference and equivalence rules do not form the basis of a construction of a new tree structure which is typically used for rendering. In the preferred method the rules are applied to the memory structure that represents the description (i.e, the DesOM) and result in changes to that structure. The role of the rules is to provide a richer description of the resource that can be exploited by applications (e.g., search engines, filter agents, etc.). This richer description does not necessarily need to be

serialised because the richer description can always be generated from the original description using the rules.

The preferred embodiment for applying the inference and equivalence rules has a limited set of actions that can be performed on the selected elements (see Rules.dtd in Section *7. Rule-based Processing using the DesOM.* This set of actions is sufficient for the Digital Video Browser System described in Section 12, however it is possible that a more extensive set of rules may be required for other applications.

Turning now to Fig. 8, there is shown a flow diagram of a preferred method of extending a description of a resource. In step 800, the method commences and a host application such as a search engine invokes a DDF processor and selects a description in response to a user request for further processing. In the next step 802, the DDF processor parses the description into a DesOM. After step 802 the method continues at step 804, where an associated set of rules are accessed using the RuleSet Attribute of the description. These set of rules may be serialised in the form of an XML document. In the next step 806, the first rule of the set is selected for processing.

The method then continues to decision block 806, where a check is made whether a pattern associated with the selected rule can be found in the DesOM. The manner in which the pattern associated with the selected rule matches a pattern in the DesOM is described in more detail in *7. Rule-based processing using the DesOM.* If the decision block 808 returns true(yes), then the processing continues at the next step 810, where the inference or equivalence action associated with the rule is initiated on the DesOM. These actions preferably initiate addition and removal of attributes and elements from the DesOM thus modifying the DesOM. Afterwards, the method selects the next rule in step A11 and the processing returns to decision block 808. If the decision block 808 returns false(no), the the processing continues at decision block 812, where a check is made whether all the selected rules have finally been processed without action. In this way, the rules are iteratively processed until no further changes can be made to the DesOM. This is advantageous in the situation where some rules are dependent on other rules.If, on the other hand, the decision block 812 returns true(yes), the processing continues at step 816 where the extended desOM is output. The method then terminates at step 818.

9. Method of Presenting Descriptions of Resources

A description could be used by many applications. Each application might exploit different properties of the description and its defining description scheme. Some of these applications will invariably need to represent description schemes and/or descriptors in a graphical or pictorial manner. For example, many descriptors could be graphically represented by icons and a user's interaction with either a description or description scheme could be mediated by icon selection.

Presentation properties for descriptors could be included as part of the description scheme however this can be non-ideal for two reasons. First, the role of the description scheme and description is to describe classes of resources and a particular resource, respectively, and it is preferable to keep both entities as concise and precise as possible. Presentation information would result in extra presentation information (e.g., icons) being part of a description scheme (and perhaps descriptions) and would therefore increase the storage and transmission costs for each description scheme. Second, different applications might prefer to present descriptions and description schemes in different ways. In other words, the presentation properties of descriptions and description schemes can be application dependent.

It is advantageous, however, to have a set of presentation rules grouped in a rule set that can be serialised, transported with and used in conjunction with the description scheme so that other applications can, if they choose to, use a similar set of presentation rules. This would not be the case if the presentation rules were tightly linked with a particular application (i.e., part of the application code base).

As with inference and equivalence rule sets, presentation rule sets can optionally be linked with a description scheme by specifying the XML document containing the presentation rule set as the value or part of the value of the ruleSets attribute in the Description element for the description scheme (see Section *3.1.2.2 Description Definition*). Presentation rule sets can be included in the ruleSets attribute along with other rule sets that might be concerned with inference and equivalence rules. In the Digital Video Browser System, which is described in Section 12, the presentation rule sets are stored with the description scheme in the ruleSets attribute. Alternatively, they could be stored with the application rather than the description scheme. Presentation rule sets stored as part of the description scheme are processed like inference or equivalence rule sets. In other words, all the rules from the individual rule sets are combined into a

single rule set. Resolution of rules is performed on the basis of rule order (as was described for inference rules in Section *8. Method of Extending Descriptions of Resources*). If an alternative method of processing presentation rule set(s) is required then the presentation rule set(s) are best stored with the application so the application can control the processing.

Presentation properties can be attributed to the descriptor definitions in a description scheme or the descriptor elements of a description using application-specific presentation rules. Unlike, inference or equivalence rules, a presentation rule is typically applied to an element definition in a DTD. Its role is to provide presentation behaviour for the instances of the descriptors defined in the description scheme. In the preferred embodiment of the Digital Video Browser System, presentation rules are only applied to descriptor definitions and not to descriptors within individual descriptions. However, it is conceivable that some applications might benefit from an ability to define presentation rules based on individual descriptors in descriptions. The rules in a presentation rule set can be formulated in a similar way to inference or equivalence rule sets.

Preferably, the Action elements of presentation rules typically involve the addition and removal of attribute definitions in element definitions (in the description scheme). Consequently the rules are targeted at element definitions rather than elements. Alternative embodiments could apply presentation rules to individual descriptions and therefore the target of these rules would be elements rather than element definitions.

Presentation rules are used in the Digital Video Browser System described in Section 12 for the following functions:

- To classify descriptors as being structural (hence belonging in a Table of Contents) or of an index nature (hence belonging to an Index);
- To assign icons to descriptors where the icons are assigned on a description scheme basis (i.e., by the addition of attribute definitions having default values to descriptor definitions), and;
- To add "Selected" attributes to all selectable descriptor definitions so that selection rules can interact with the presentation of the descriptions (e.g., so the application can differentiate visually between selected and non-selected descriptors).

The preferred method involves associating a set of rules with a description scheme that can influence the presentation properties of descriptors in descriptions which are

conformant with a particular description scheme. It is an advantage to have these presentation rules grouped in a rule set that is either linked to a description scheme so that applications can utilise the defined set of presentation properties if required. Alternatively an application can select to use its own set of presentation rules.

Turning now to Fig. 9, there is shown a flow diagram of a preferred method of visually presenting a description of a resource. In step 900, the method commences and a host application such as a search engine invokes a DDF processor. In the next step 901, a description is selected for presentation. This selection can occur by way of user input or by way of another application. The method then continues at step 902, where the associated defining description scheme is read into memory. The description scheme in memory comprises an array of element defintion where each element definition has an array of attribute definitions. Alternatively, the DDF processor can parse the description into a DesOM. After step 902 the method continues at step 904, where the presentation set of rules are accessed using the RuleSet Attribute of the description. In the next step 906, the first presentation rule of the set is selected for processing.

The method then continues to decision block 908, where a check is made whether a pattern associated with the selected rule can be found in the DesOM. A pattern matching process similar to that described in *7. Rule-based processing using the DesOM* would be suitable. If the decision block 908 returns true(yes), then the processing continues at the next step 910, where the the attribute definition(s) associated with the rule is removed or added to the array in memory. Afterwards, the method selects the next rule in step 911 and the processing returns to decision block 908. If the decision block 908 returns false(no), the processing continues at decision block 912, where a check is made whether all the selected rules have finally been processed without action. In this way, the rules are iteratively processed until no further changes can be made to the array in memory. This is advantageous in the situation where some rules are dependent on other rules. If, on the other hand, the decision block 912 returns true(yes), the processing continues at step 916 wherein a modified description is created using said modified as a template. This modified description is then output to an output device. For example, the modified description and it's associated resources, such as digital video resources or DVDs, can be rendered on a display or a printing device.

<u>10. Method of Selecting Resource Descriptions</u>

Selection rules can be used to formulate queries directed at collections of descriptions (e.g., digital libraries). A query can be viewed as a request to select those descriptions or components of descriptions (i.e., descriptors) that match a specified pattern. Like inference and equivalence rules, selection rules are typically directed at elements rather than element definitions. Unlike inference, equivalence and presentation rules, however, selection rules may be generated on a one-off basis and not collected in rule sets that are serialised in an XML document. For example, a query is usually formulated with help from the user, then processed, and the results presented to the user for their evaluation.

Selection rules often depend on presentation rules in that the selection action must be able to be interpreted by the application and presented to the user. For example, a selection action could simply set a (presentation) attribute for descriptors that match the specified pattern.

Selection rules are typically associated with the application. In the preferred embodiment of the Digital Video Browser System, selection rules use the same grammar as all other rules (see Section *7. Rule-based Processing using the DesOM*). However, typically the only Action that is invoked by a selection rule is the Select action. Consequently it would be possible to define a more specific grammar for selection rules (e.g., SelectionRules.dtd having just a Select action being allowed).

The Select action of a selection rule has three attributes which specify how the selection action is implemented. The value of the attribute attName refers to the attribute name used for a descriptor that is able to represent the action of being selected. This attribute would typically have been generated using a presentation rule. If the element matched by the pattern does not contain such an attribute, then the selection process will search for ancestors of the matched element in the DesOM (i.e., up the description tree) until it locates an element with the specified attribute name. In the above DTD this attribute name is provided with a default value of "selected". The value of the second attribute attValue refers to the value that the "selected" attribute should be assigned in order to indicate selection. The DTD also provides a default value of "YES". The third attribute specifies whether all selectable ancestors should also be selected. So, for example, if a user selects a Shot descriptor because of a matched descriptor contained in

the Shot descriptor, then the user should also select the ancestors of the Shot descriptor (i.e, the Scene descriptor and the VideoClipDescription descriptor).

In this way, the Select element provides information to the application on which elements have matched the specified pattern in the selection rule. Clearly the application needs to be aware of the attribute used to provide this information, hence the interaction between presentation and selection rules. In the Digital Video Browser System (see Section 12), selection rules are used to implement searches in a Digital Video Library.

The preferred method involves that of representing queries by selection rules which attempt to find matches to a rule's specified element pattern. The "select" action that is executed on a successful pattern match typically modifies attributes established by presentation rules, so that the selection process can interact with the application.

Turning now to Fig. 10, there is shown a flow diagram of a preferred method of selecting one or more descriptions or part of one or more descriptions of a resource. In step 1000, the method commences and a host application such as a search engine invokes a DDF processor. In the next step 1002, a user inputs a query which is formulated as a rule in step 1004.The search engine then selects in step 1005 a first description for evaluation. The method then continues at step 1006, where the DDF processor parses the description into a DesOM.

The method then continues to decision block 1008, where a check is made whether a pattern associated with the selected rule can be found in the DesOM. The manner in which the pattern associated with the selected rule matches a pattern in the DesOM is described in more detail in *7. Rule-based processing using the DesOM.* If the decision block 1008 returns true(yes), then the processing continues at the next step 1010, where the select action associated with the rule is initiated on the DesOM. The details of the select action is described above. Afterwards, the method then continues at decision block 1012 where a check is made whether the last description has been searched. If the decision block returns false(no) the processing continues at step 1014 where the next description is selected. Otherwise, the processing continues at step 1016, where the results of the searching process is output. The method then terminates at step 1018.

11. Method of Translating Descriptions of Resources

Often descriptions of resources will be in a language different from the request. Rather than store copies of the descriptions in each language, the preferred method stores

only one copy of the descriptions in one language. Preferably, the language is English. The preferred method is then provided with a number of rule sets that enable the translation of the descriptions to the language of the request. For example, the description may have a "color" attribute and a color attribute value "red". If the request is received in French, then the preferred method will translate the description to French. In the example given, "color" and "red" will be translated to their French equivalent. This is a form of inter-language equivalence. This procedure is similar to the way Inference Rules are processed, but on a conditional basis. Inference rules are preferably not processed on a conditional basis as described here for translation rules.

Turning now to Fig. 11, there is shown a flow diagram of a preferred method of translating a description of a resource. In step 1100, the method commences and a host application such as a search engine invokes a DDF processor and selects a description in response to a user request for further processing. In the next step 1102, the DDF processor parses the description into a DesOM. After step 1102 the method continues at decision block 1103, where a check is made whether the language of the request is different from the language of the description. This check is accomplished by comparing the language attributes of both the request and the description.

If the decision block 1103 returns true(yes), the processing continues at step 1104, where an associated translation set of rules are accessed using the RuleSet Attribute of the description. These translation set of rules may be serialised in the form of an XML document. On the other hand, if the decision block returns false(no) then the processing continues at step 1116. After completion of step 1104, the method continues at step 1106, where the first rule of the set is selected for processing.

The method then continues to decision block 1106, where a check is made whether a pattern associated with the selected rule can be found in the DesOM. The manner in which the pattern associated with the selected rule matches a pattern in the DesOM is described in more detail in *7. Rule-based processing using the DesOM.* If the decision block 1108 returns true(yes), then the processing continues at the next step 1110, where the translation action associated with the rule is initiated on the DesOM. These actions initiate the removal and addition of attributes and elements from the DesOM. The removal and addition action substitutes the language of the attributes and elements for another. Afterwards, the method selects the next rule in step B11 and the processing

returns to decision block 1108. If the decision block 1108 returns false(no), the the processing continues at decision block 1112, where a check is made whether all the selected rules have finally been processed without action. If, on the other hand, the decision block 1112 returns true(yes), the processing continues at step 1116 where the extended desOM is output. The method then terminates at step 1118. Alternatively it is also possible to include an action of a rule which invokes a DescriptorHandler method to translate the content of the selected Descriptor.

12. Digital Video Browser System

The functionality of the Digital Video Browser System, that is described in this Section, is enabled by the descriptions of digital video that are automatically generated using a description scheme, designed for digital video resources, such as that included in Appendix D.

The Digital Video Browser System allows a user to browse the digital video in a non-linear manner, manually annotate the digital video to provide additional descriptive information that was not able to be automatically generated, and to search for the presence of various descriptors in a description. It should be clear to the reader that all this functionality is enabled by an interaction of the user with the description scheme and the individual descriptions of the digital video resources and that the browser that is described in the following section can in essence be applied to any other electronically-accessible resource.

A typical Digital Video Browser System is shown in Fig. 12. The system contains a Video Browser Panel 1200 which consists of a Viewing Panel 1201, a Table-of-Contents (or TOC) Panel 1202, and an Index Panel 1203. Outside of the Video Browser Panel 1200 but within the system are three buttons required for user interaction; a Search button 1205, a Play button 1206, and an On/Off button 1207.

User interaction with the panels of the Digital Video Browser System can be mediated by a touch-sensitive Video Browser Panel, however this feature is not necessary for the operation of the system. The operation of the Digital Video Browser System will now be discussed in the terms of Fig. 12.

When a new digital video resource is added to the Digital Video Browser System a predetermined description scheme is applied to the digital video resource resulting in the content creation methods of the relevant descriptor handlers in the description schemes

being initiated. Other implementations might provide more than one description scheme which can be applied to the digital video resources. For example, a Digital Video Browser System might provide the description schemes contained in Appendices B and D. In such an embodiment the user would require a means to select the description scheme that he/she would like to apply to each new digital video resource. So, for example, if he/she was adding a new digital video resource containing the footage from a football match then he/she would most likely use the description scheme in Appendix B, however if the digital video resource contained some footage of a recent holiday, then it's likely that the description scheme contained in Appendix D would be more appropriate.

If more than one description scheme is available then the selection of the most appropriate description scheme to use could also be automated to some extent. The resource to be described could be analysed to see if it contained key features that typically indicate the use of a particular description scheme. For example, the sound track of a digital video resource could be analysed for repetitive whistle sounds arising from a referee's whistle. If detected, such sounds could provide evidence for the use of a particular description scheme (e.g., the description scheme shown in Appendix B).

In a simple description scheme such as that included in Appendix B there is a single descriptor handler specified for the description (which is also a descriptor), which generates the entire content for the description.

In other description schemes, more than one descriptor may have an associated descriptor handler which is responsible for automatically generating the content of just that descriptor. For example, consider the description scheme shown in Appendix D. The VideoDescription descriptor D1 has an associated descriptor handler D2 which provides a method to automatically segment the digital video resource into a series of individual shots. The Shot descriptor D3 has an associated descriptor handler D4 which provides a method to automatically select a key frame from a specific shot and then generate a series of semantic labels which provide some information about the content of the particular shot (e.g., whether or not the shot contained people, was an indoors or outdoors shot, etc.). These descriptor handler methods are executed on the creation of a descriptor in the description being generated. Therefore the description can be progressively constructed using the description scheme (effectively as a template) and the set of descriptor handlers

that provide the methods for automatically generating the content for their relevant descriptors. An example of such a generated description is provided in Appendix E.

In the case of the Digital Video Browser System depicted in Fig. 12, the descriptors able to be accessed in the Index Panel, rather than the TOC Panel are classified as Index Descriptors. The classification of descriptors as Index or TOC descriptors is achieved using presentation rules (see Section *8. Method of Presenting Descriptions of Resources*), with each description scheme being used by the Digital Video Browser System having a corresponding presentation rule set. For example, a presentation rule could be applied to each of the descriptor definitions in the description scheme to add an attribute definition to the descriptor's definition for the purposes of this classification. The added attribute definition could have a attribute default of #FIXED "Index" or #FIXED "TOC" to classify an Index and TOC descriptor, respectively. [Note: The use of the #FIXED keyword in the default value means that changing the value of the classifier from its default value results in an invalid XML construct and hence an invalid description.]

Selecting which descriptors are to be used as Index descriptors is similar to selecting which key words or phrases you would include in the index of a book. In other words, it is an authoring task that results in presentation rules. In general, a descriptor that is classified as a TOC descriptor represents a structural element of the resource (i.e., a component that would normally appear in the TOC of a book). So, for example, a Shot descriptor is a TOC descriptor. An Index descriptor typically represents a property of a TOC descriptor (e.g., a Shot descriptor could contain people scenes, be an indoor or outdoor scene, etc.).

The Index descriptors are the leaf nodes of the internal tree structure used to represent the description [The internal representation of descriptions is discussed in detail in Section *2.3 Description Object Model (DesOM)*]. In the absence of presentation rules, this property can also be used to implicitly differentiate between Index and TOC descriptors in an implemented Digital Video Browser System. In the preferred embodiment of the Digital Video Browser System, explicit differentiation between Index and TOC Descriptors is achieved using presentation rules. A set of presentation rules applicable to the description scheme in Appendix D is shown in Appendix F.

The Digital Video Browser System has access to a collection of digital video resources, which is hereinafter referred to a Digital Video Library. A newly described

digital video resource can be simply appended to an existing collection of described digital video resources. Alternatively (see Section *11. Remote Digital Video Browser Devices*), the user can insert a new item at the desired location using a drag-and drop means. The Digital Video Library is itself a resource able to be described. Therefore, on initialising the Digital Video Browser System a description scheme for a Digital Video Library is used to automatically generate a description for the Digital Video Library.

The description of the Digital Video Library can be very simple containing just a hierarchical representation of the individual descriptions of digital video resources described in the library. In other words, the description need not know about the location of the digital video resources described in the library. It is merely a catalogue of the descriptions of the digital video resources stored in the library. Each individual description has a reference to its corresponding digital video resource.

An example of a description scheme for a Digital Video Library is included in Appendix G. The Digital Video Library's description can contain zero or more Section elements or zero or more Item elements, where each Item element refers to an individual description in the Digital Video Library (i.e., an XML document). A description of a Digital Video Library conforming to the description scheme included in Appendix G is shown in Appendix H.

During browsing the user can select sections of Digital Video Library by selecting the relevant descriptors in the TOC Panel 1202 in the Video Browser Panel 1200. This selection method provides non-linear access to the digital video resource(s). Typically these selections are highlighted in the TOC panel to indicate which are currently selected. The user can choose to play all the highlighted selections by pressing the "Play" button 1206.

Alternatively the user can search for sections, items or parts of items of the Digital Video Library by selecting relevant Index descriptors in the Index Panel 1203. In a simple Digital Video Browser System implementation, the Index descriptors might imply simple boolean presence of a specified feature. For example, the PeopleScene Index descriptor (see D5 in Appendix D) could indicate whether people are either present or absent from the shot. In a more sophisticated Digital Video Browser System the Index descriptors might require some representative value. For example, a "Date" Index descriptor would require a specified value before a search could be performed.

Searches can be performed within a TOC context in the Digital Video Library. For example, if a user wanted to search for PeopleScene descriptors within a specific digital video resource, the user could select the TOC descriptor for that particular resource in the TOC Panel 1202 and then select the desired Index descriptor in the Index panel 1203 and press the "Search" button 1205 in the Digital Video Browser System. The search process would then result in all TOC descriptors that satisfied the search criteria becoming selected (e.g., highlighted) in the TOC Panel 1202. The user could then select to play all the selected sections of the digital video resource by pressing the "Play" button 1206.

Searches can be implemented in the Digital Video Browser System using selection rules (see Section *10. Method of Selecting Resource Descriptions*). The TOC context is automatically inserted as part of the pattern of the selection rule. The search process applies the selection rule pattern to each relevant description and updates a selection attribute that has been added for all selectable attributes using a presentation rule. Selectable attributes will vary between description scheme and application. In the case of the description scheme included in Appendix D the only descriptors that might be classified as selectable would be the VideoDescription and Shot descriptors (see the presentation rules in Appendix F).

The Digital Video Browser System also provides functionality for manual annotation, in conformance with the description scheme, of a digital video resource. If a particular TOC descriptor is selected, then the relevant Index descriptors 1209 can be displayed in the Index Panel 1203. The Index descriptors are preferably represented by icons (which in the preferred embodiment are specified by presentation rules targeted at the descriptor definitions). The selected TOC descriptor can be viewed (played) and then manually annotated by dragging icons representing the Index descriptors (e.g., 1210) into an Annotation Region 1204 of the Viewing Panel 1201. Annotations created in this fashion are then added to the description of the resource and are available for subsequent browsing.

Annotations in the form of titling various TOC Descriptors could also be possible in some implementations of a Digital Video Browser System. For example, in a Digital Video Browser System implemented in software on a regular personal computer, the screen representation of the Descriptor could be selected and then the title for the descriptor could be entered using the computer's keyboard. In alternative embodiments,

in which access to the Video Browser Panel 1200 is provided via a touch-sensitive display, user entry of textual titles could be mediated by a pen interface or via a method whereby a particular descriptor is selected by touch, and the title communicated by the user speaking the title words and a speech-to-text module in the Video Browser System converting the spoken words to text and displaying the result where a title is expected on the display.

Whenever new descriptions are retrieved for browsing the description is processed into a DesOM. Before the description is actually presented in the Video Browser System, any inference or equivalence rules (see Section *8. Method of Extending Descriptions of Resources*) that are associated with the description's description scheme are processed. This processing involves iterating through the defined inference rules until no more changes can be made to the description. Clearly, this rule processing requires that there are no circular dependencies in the rule set. The inference and/or equivalence rules will result in the creation of new descriptors which have been inferred from those that were part of the serialised description. Preferably, any new descriptors created by this process will have been defined as part of the relevant description scheme (and as such will have been classified as an Index or TOC descriptor). The inference rules will need to be reprocessed in the event of any annotations being created.

13. Remote Digital Video Browser Devices

The Digital Video Browser System described in the previous section can also be implemented as a dedicated remote device. In this section two possible remote device embodiments of the Digital Video Browser System are described with respect to Fig. 13 and Fig. 14.

The first remote device of the Digital Video Browser System is shown in Fig. 13. In this embodiment the Video Browser 1300, contains no storage for the Digital Video Library. The Video Browser 1300 communicates with a Server 1310 using a wireless transmitter/receiver 1302 and a wireless connection 1303. The Server 1310 has a connection 1313 with a storage device that contains the Digital Video Library 1311. All the digital video resources that can be browsed by the Video Browser are stored in this Digital Video Library. Preferably, in this remote device all the descriptions of the digital video resources are also stored in this library 1311. The Server 1310 also has a connection 1314 to a large display 1312 that can be used for public viewing of the digital video

resources. Preferably, the connections between the Server 1310 and the Digital Video Library 1311 and between the Server 1310 and the large display 1312 are wired connections.

New digital video resources can be added to the Digital Video Library 1311 which is directly connected to the Server 1310 independently of the Video Browser device 1300. As the resources are added to the Digital Video Library 1311 (from, for example, a digital video camera), descriptions for the digital video resources are automatically generated using the description scheme. Also at this time, usually after the description has been generated, the user could optionally title sections of the digital video resource. These titles would then be visible when browsing using the Digital Video Browser device.

On power-up the Video Browser device connects to the Server 1310 using the wireless connection 1303. The Server 1310 communicates to the Digital Video Browser device a description of the Digital Video Library. This description, like descriptions of the digital video resources, conforms to a description scheme (in this case for a Digital Video Library), and is serialised in an XML document. An example of a description of a Digital Video Library is shown in Appendix H.

The remote Digital Video Browser device 1300 can either store the relevant description schemes permanently, or download these description schemes at the time of making its connection with the Server 1310. The latter method of obtaining the description schemes is the preferred method. The description of the Digital Video Library and the relevant description schemes contain all the information required to display an Index and TOC panel on the Digital Video Browser device 1300. The user can then use the Digital Video Browser device to navigate through the Digital Video Library, selecting or searching for video resources to view. Prereably, the navigation through the TOC and Index panels is enabled via a touch-sensitive screen. Other methods of navigation (e.g., a pen or simple keyboard) could also be used.

Only when a Digital Video Browser user selects to "Play" a particular selection of digital video resources, is it necessary to transmit the required digital video resources from the Digital Video Library 1311 to the remote Digital Video Browser device 1300. Preferably the digital video resources are stored and transmitted in compressed form (e.g., MPEG-1 or MPEG-2) therefore minimising the bandwidth of the required wireless

connection 1303 between the Server 1310 and the remote Digital Video Browser device 1300.

The remote Digital Video Browser device can optionally have an additional button (to those shown in Fig. 12), which can be used to direct the Viewing Panel 1301 of the remote Digital Video Browser device to a large display 1312 connected to the Server 1310. This redirection can be achieved by transmitting a description of the required presentation (i.e., an XML document) from the remote Digital Video Browser device 1300 to the Server 1310. This description would conform to a Video Presentation Description Scheme that could be as simple as just a list of all the selected sections of the selected digital video resources. This description would be interpreted by the Server 1310 and the corresponding sections of the selected digital video resources would be rendered to the large display 1312. Preferably the rendering is performed by the Server 1310 and pixel data would be transmitted over the connection 1314, however if the large display 1312 had the processing ability to decode the compressed digital video resource, then the compressed resource could be transmitted over the connection 1314 and then decoded and rendered in the large display 1312.

Clearly, presentation rules could be applied to the presentation of the selected items in the same way as presentation rules are applied to a description of a digital video resource. Some presentation rules that could be applicable to the presentation of digital video resources include rules that specify the type of transitions to be inserted between shots of a particular digital video clip (e.g., fades, cuts, wipes, etc.) and whether clip titles are to be rendered over the presented video and the style of title rendering to be used. These rules could be collected in a presentation rule set that is linked with the Video Presentation Description Scheme in the same way that sets of presentation rules could be linked to the Digital Video Resource Description Scheme (see Appendix D).

Alternative Digital Video Browser implementations could allow users to specify additional presentation rules for the presentation of selected digital video resources. For example, an implementation could allow a user to specify whether a particular selection was to be played at recorded, slow or fast speed. Altering the speed of video playing can provide interesting presentation effects. Similarly, the Digital Video Browser user might also be able to specify the types of transitions to use on a one-off presentation basis rather than a default basis as provided by rules linked to the Video Presentation Description

Scheme. These one-off presentation rules can be combined into a single rule set which is referenced by the Description element of the presentation description that is communicated to the Server 1310 when the user chooses to play the selected digital video resources (whether on the Digital Video Browser device itself or, more likely, when the presentation has been re-directed to the large display 1312).

An example of a Video Presentation Description Scheme, which could be used with the Video Description Scheme shown in Appendix D, is shown in Appendix I. In this description scheme, a standard set of presentation rules is provided as part of the description scheme. These rules have been collected into a rule set and stored in the XML document which, in the case of the example is called "VideoPresentationRules.xml". The rule set has then been referenced by the description scheme by specifying an ENTITY for the ruleSets attribute I1 of the VideoPresentationDescription element. The attribute userPresentationRules I2 has been added to the VideoPresentationDescription subclass of the Description element to be able to contain an ENTITY that specifies an xml document that contains any presentation-specific rules.

An example of a video presentation description that conforms to the Video Presentation Description Scheme, which is included in Appendix I, is shown in Appendix J. A set of presentation-specific rules has been specified for the particular presentation using the userPresentationRules attribute of the VideoPresentationDescription element (see J1). Clearly the example description scheme and presentation description included in Appendices I and J pertain to the Video Description Scheme included in Appendix D since they refer to particular descriptors in that description scheme. For example, the VideoDescriptionReference element contains zero or more references to Shot elements in the referenced video descriptions. In particular the shotIDRef element J2 specifies a particular shot descriptor in the description contained VideoEg1.xml, by using a reference to the ID of that descriptor in the description. It is not necessary to use a Video Presentation Description Scheme that is directed so specifically at a particular description scheme. For example, if a Digital Video Browser System was implemented with more than one description scheme, then a more general Video Presentation Description Scheme can be used.

The ability to be able to re-direct the Viewing Panel 1301 to a large display 1312 connected to the Server 1310 is a useful feature as the user can select the sections of

his/her Digital Video Library that he/she wishes to share with an audience using the remote Digital Video Browser device. That selection can then simply be played to the large display 1312.

A second remote device implementation of the Digital Video Browser System is shown in Fig. 14. In this implementation the Digital Video Browser 1400 is implemented as a remote device that has a capability to read Digital Video Disks (DVDs). Typically each DVD is treated like an independent Digital Video Library and consequently each DVD has its own description of the Digital Video Library contained on the DVD. When the DVD 1415 is inserted into the remote Digital Video Browser device 1400 the Video Browser 1400 reads the description of the Digital Video Library contained on the DVD. In this device the description scheme required to interpret the Digital Video Library would preferably reside in the remote Digital Video Browser device, however it is conceivable that the description scheme could also be located on each DVD. Similarly the description schemes required to interpret the descriptions of the digital video resources could either be located on the DVD or in the remote Digital Video Browser device. In the preferred implementation of this device, all the required description schemes are located in the remote Digital Video Browser device 1400. New description schemes for digital video resources can be downloaded via the wireless transmitter/receiver 1402 and wireless connection 1404 to a server or computer 1413 connected to a network 1414. Alternatively, the remote Digital Video Browser can be docked at a server or networked computer for the download of new description schemes.

Once the description of a Digital Video Library has been read from the DVD 1415 then the user can navigate through this Digital Video Library as described previously. Sections of described digital video resources can be selected and played on the remote device. The device is in many respects very similar to the device depicted in Fig. 13, with the exception that it does not require a Server to store the digital video resources and descriptions.

Sections of the selected digital video resources can be selected for viewing on a large display 1410 that has a wireless connection 1403 with the remote Digital Video Browser device. This large display 1410 must either contain, or be directly connected to, a processor able to decode and render the compressed digital video resource that is transmitted via the wireless connection 1403. As with the remote device depicted in Fig.

13, a description of the required presentation is communicated to the large display 1410. In addition, any digital video resources required for the presentation description to be rendered must also be communicated. These resources are typically communicated in compressed (encoded) form (e.g., MPEG-1 or MPEG-2). The processor either contained in, or directly connected to, the large display 1410 renders the presentation using the presentation description and its associated digital video resources. The rendering process can typically adapt to the resolution of the large display 1410, which is usually greater than that of the handheld device.

In the preferred implementation of this device 1400, if the description of the required presentation requires that only particular sections of a selected digital video resources be presented, then these required sections can be isolated from the original digital video resource, recoded if necessary in the handheld device, and then communicated to the large display 1410. This approach reduces the communication bandwidth of the wireless connection 1403. Alternatively, the entire digital video resource can be communicated and the processor that renders the presentation will need to extract the relevant sections of the digital video resource(s). The latter implementation is more costly in bandwidth but does not involve recording of digital video resources in the remote device.

In order to facilitate resource discovery on different DVDs, this remote Digital Video Browser device can also have an ability to generate printed DVD covers that display the contents of the DVD in a graphically pleasing manner. This facility can be achieved using a wireless connection 1405 to either a printer with some processing ability 1412, or to a computer directly connected to a printer (not shown in Fig. 14). Typically the Digital Video Browser device would send to the printing device (1412 or the computer directly connected to a printer), a description of the (printed) presentation that is to be the printed DVD cover.

Description schemes for this presentation could be designed just as they can be designed for video presentations (e.g., see Appendix I). For example, at the simplest level the Digital Video Library description could form the basis of the printed presentation. Presentation rules could then be used to specify the spatial layout and colour arrangement of the printed presentation, and also the association of icons or key frames to particular descriptors in the description. The presence of visual reminders of the content of the

DVD, such as icons or key frames, are important for purposes of identification and retrieval.

A processor, which is located either in the printer 1412 or in a computer connected to the printer could then use the description of the required printed presentation and any provided presentation rule sets to render a DVD cover for the particular DVD using the provided key frames. This processor would need to be able to interpret the description of the printed presentation.

14. Alternative Preferred Embodiment of Apparatus

The methods of Figs. 1A,1B,2B and 7A to 11 can be practiced using a conventional general-purpose computer, such as the one shown in Fig. 15 wherein the processes of Figs. 1A,1B,2B and 7A to 11 may be implemented as software executing on the computer. In particular, the method steps are effected by instructions in the software that are carried out by the computer. The software may be divided into two separate parts; one part for carrying out the processing steps; and another part to manage the user interface between the latter and the user. The software may be stored in a computer readable medium, including the storage devices described below, for example. The software is loaded into the computer from the computer readable medium, and then executed by the computer. A computer readable medium having such software or computer program recorded on it is a computer program product. The use of the computer program product in the computer preferably effects an advantageous apparatus in accordance with the embodiments of the invention.

The computer system 1500 consists of the computer 1502, a video display 1516, and input devices 1518, 1520. In addition, the computer system 1500 can have any of a number of other output devices including line printers, laser printers, plotters, and other reproduction devices connected to the computer 1502. The computer system 1500 can be connected to one or more other computers via a communication interface 1508b using an appropriate communication channel 1530 such as a modem communications path, a computer network, or the like. The computer network may include a local area network (LAN), a wide area network (WAN), an Intranet, and/or the Internet

The computer 1502 itself consists of a central processing unit(s) (simply referred to as a processor hereinafter) 1504, a memory 1506 which may include random access memory (RAM) and read-only memory (ROM), input/output (IO) interfaces 1508a,

1508b & 1508c, a video interface 1510, and one or more storage devices generally represented by a block 1512 in Fig. 15. The storage device(s) 1512 can consist of one or more of the following: a floppy disc, a hard disc drive, a magneto-optical disc drive, CD-ROM, magnetic tape or any other of a number of non-volatile storage devices well known to those skilled in the art. Each of the components 1504 to 1512 is typically connected to one or more of the other devices via a bus 1514 that in turn can consist of data, address, and control buses.

The video interface 1510 is connected to the video display 1516 and provides video signals from the computer 1502 for display on the video display 1516. User input to operate the computer 1502 can be provided by one or more input devices 1508b. For example, an operator can use the keyboard 1518 and/or a pointing device such as the mouse 1520 to provide input to the computer 1502.

The system 1500 is simply provided for illustrative purposes and other configurations can be employed without departing from the scope and spirit of the invention. Exemplary computers on which the embodiment can be practiced include IBM-PC/ATs or compatibles, one of the Macintosh ™ family of PCs, Sun Sparcstation ™, or the like. The foregoing are merely exemplary of the types of computers with which the embodiments of the invention may be practiced. Typically, the processes of the embodiments, described hereinafter, are resident as software or a program recorded on a hard disk drive (generally depicted as block 1512 in Fig. 15) as the computer readable medium, and read and controlled using the processor 1504. Intermediate storage of the program and pixel data and any data fetched from the network may be accomplished using the semiconductor memory 1506, possibly in concert with the hard disk drive 1512.

In some instances, the program may be supplied to the user encoded on a CD-ROM or a floppy disk (both generally depicted by block 1512), or alternatively could be read by the user from the network via a modem device connected to the computer, for example. Still further, the software can also be loaded into the computer system 1500 from other computer readable medium including magnetic tape, a ROM or integrated circuit, a magneto-optical disk, a radio or infra-red transmission channel between the computer and another device, a computer readable card such as a PCMCIA card, and the Internet and Intranets including email transmissions and information recorded on websites and the like. The foregoing are merely exemplary of relevant computer readable mediums. Other

computer readable mediums may be practiced without departing from the scope and spirit of the invention.

The preferred methods of the invention may alternatively be implemented in dedicated hardware such as one or more integrated circuits performing the functions or sub functions of preferred methods of the invention. Such dedicated hardware may include graphic processors, digital signal processors, or one or more microprocessors and associated memories.

The foregoing only describes a small number of embodiments of the present invention, however, modifications and/or changes can be made thereto by a person skilled in the art without departing from the scope and spirit of the invention. The present embodiments are, therefore, to be considered in all respects to be illustrative and not restrictive. In the context of this specification and accompanying aspects of invention, the word "comprising" means "including principally but not necessarily solely". Variations of the word comprising, such as "comprise" and "comprises" have correspondingly varied meanings.

The following numbered paragraphs set forth aspects of the invention, including

1. A method of generating a printed presentation based on a description of the digital resources, contained on a medium accessed by a source processing device, on a destination printing device, said method comprising the steps of:

providing a description scheme for the class of presentations using a declarative description definition language which contains definitions for descriptor components of the description scheme, wherein each said descriptor component comprises the association of a resource attribute with a representative value for that attribute;

associating with the said description scheme a set of presentation rules which specify characteristics of the style of the presentation for descriptions created using the said description scheme;

creating a description of the said printed presentation using the said description scheme as the template for the description's instantiation in the source processing device;

generating the printed presentation from the said description and any associated image content on the said destination printing device.

2. The method according to paragraph 1 wherein said description scheme can have an associated reference to procedural code for the instantiation of a descriptor in the description of the presentation.

3. The method according to paragraphs 1 and 2 wherein said presentation rules can specify the spatial layout of the printed presentation.

4. The method according to paragraphs 1 to 3 wherein said presentation rules can specify colour information for the printed presentation.

5. The method according to paragraph 4 wherein the said associated image content contains image frames from the digital video resources stored on the DVD.

6. The method according to paragraph 5 wherein the said destination printing device is connected to the source-processing device via a wireless connection.

7. The method according to paragraph 6 wherein the said destination printing device contains a processor capable of rendering the presentation from the description and the associated image content.

8. The method according to paragraph 5 wherein the source processing device is connected via a wireless connection to a another processor which can render the printed

presentation from the description and the associated image content, and communicate the rendered presentation to the destination printing device.

9. The method according to paragraph 1, further comprising associating with the said description a further set of presentation rules which specify characteristics of the style of the presentation to be generated from the said description.

10. An apparatus for implementing any one of the aforementioned methods.

11. A computer program product including a computer readable medium having recorded thereon a computer program for implementing any one of the methods described above.

12. A method as substantially described with reference to the accompanying drawings.

Dated this Twenty Ninth Day of January 1999

Canon Kabushiki Kaisha

Patent Attorneys for the Applicant

SPRUSON & FERGUSON

Appendix A: Core DDF Element Definitions

Core.ddf

```
<!-- Core.ddf: Contains the definitions of core DDL elements -->
<!----------------------------------------------------------------------------->

<!-- Include commonly used NOTATIONS here -->
<!---------------------------------------------------------->
<!NOTATION JavaClass SYSTEM "java">

<!-- Include commonly used ENTITIES here -->
<!------------------------------------------------------->
<!ENTITY % DataTypes "(Int | Float | Double | String | Date | Time | ID | IDREF |
IDREFS | ENTITY | ENTITIES)">

<!-- Definition of core elements -->
<!-------------------------------------------->
<!ELEMENT Descriptor (ANY)>
<!ATTLIST Descriptor
  id                        ID              #IMPLIED
  xml:lang                  CDATA           "en"
  dataType                  %DataTypes;     "String"
  superElement              NMTOKEN         #IMPLIED
  handler                   ENTITY          #IMPLIED
>
<!ELEMENT Description (Descriptor+)>
<!ATTLIST Description
  superElement              NMTOKEN         #FIXED "Descriptor"
  resource                  ENTITY          #REQUIRED
  dateResourceLastModified  CDATA           #IMPLIED
  ruleSets                  ENTITIES        #IMPLIED
>

<!-- Definition of selected relationship elements -->
<!------------------------------------------------------------>
<!ELEMENT ParallelSequence (Descriptor+)>
<!ATTLIST ParallelSequence
superElement                NMTOKEN         #FIXED "Descriptor"
>
<!ELEMENT SerialSequence (Descriptor+)>
<!ATTLIST SerialSequence
superElement                NMTOKEN         #FIXED "Descriptor"
>
<!ELEMENT Neighbours (#PCDATA)>
<!ATTLIST Neighbours
superElement                NMTOKEN         #FIXED "Descriptor"
dataType                    %DataTypes;     #FIXED "IDREFS"
```

```
>
<!ELEMENT Before (#PCDATA)>
<!ATTLIST Before
    superElement        NMTOKEN         #FIXED "Descriptor"
    dataType            %DataTypes;     #FIXED "IDREFS"
>
<!ELEMENT After (#PCDATA)>
<!ATTLIST After
    superElement        NMTOKEN         #FIXED "Descriptor"
    dataType            %DataTypes;     #FIXED "IDREFS"
>
<!ELEMENT InFrontOf (#PCDATA)>
<!ATTLIST InFrontOf
    superElement        NMTOKEN         #FIXED "Descriptor"
    dataType            %DataTypes;     #FIXED "IDREFS"
>
<!ELEMENT Behind (#PCDATA)>
<!ATTLIST Behind
    superElement        NMTOKEN         #FIXED "Descriptor"
    dataType            %DataTypes;     #FIXED "IDREFS"
>

<!-- Definition of link elements -->
<!---------------------------------------->
<!ELEMENT CLink (#PCDATA)>
<!ATTLIST CLink
 superElement           NMTOKEN         #FIXED "Descriptor"
dataType                %DataTypes;     #FIXED "IDREF"
>
<!ELEMENT ILink (#PCDATA)>
<!ATTLIST ILink
 superElement           NMTOKEN         #FIXED "Descriptor"
    dataType            %DataTypes;     #FIXED "IDREFS"
>

<!-- Definition of locator and extent elements -->
<!-------------------------------------------------------->
<!ELEMENT Locator (Extent+)>
<!ATTLIST Locator
    superElement        NMTOKEN         #FIXED "Descriptor"
 resource               ENTITY          #REQUIRED
>
<!ELEMENT Extent (Descriptor+)>
<!ATTLIST Extent
    superElement        NMTOKEN         #FIXED "Descriptor"
>
<!ELEMENT ImageExtent (Descriptor+)>
```

A3

```
<!ATTLIST ImageExtent
 superElement                NMTOKEN            #FIXED "Extent"
>
<!ELEMENT RectImageExtent (RectImageExtentX0, RectImageExtentY0,
        RectImageExtentHeight, RectImageExtentWidth)>
<!ATTLIST RectImageExtent
superElement              NMTOKEN              #FIXED "ImageExtent"
 "
>
<!ELEMENT RectImageExtentX0 (#PCDATA)>
<!ATTLIST RectImageExtentX0
 superElement             NMTOKEN              #FIXED "Descriptor"
 dataType                 %DataTypes;          #FIXED "Int"
>
<!ELEMENT RectImageExtentY0 (#PCDATA)>
<!ATTLIST RectImageExtentY0
 superElement             NMTOKEN              #FIXED "Descriptor"
 dataType                 %DataTypes;          #FIXED "Int"
>
<!ELEMENT RectImageExtentHeight (#PCDATA)>
<!ATTLIST RectImageExtentHeight
 superElement             NMTOKEN              #FIXED "Descriptor"
 dataType                 %DataTypes;          #FIXED "Int"
>
<!ELEMENT RectImageExtentWidth (#PCDATA)>
<!ATTLIST RectImageExtentWidth
 superElement             NMTOKEN              #FIXED "Descriptor"
 dataType                 %DataTypes;          #FIXED "Int"
>

<!ELEMENT VideoExtent (VideoExtentStart, VideoExtentEnd, ImageExtent?)>
<!ATTLIST VideoExtent
 superElement             NMTOKEN              #FIXED "Extent"
>
<!ELEMENT VideoExtentStart (#PCDATA)>
<!ATTLIST VideoExtentStart
 superElement             NMTOKEN              #FIXED "Descriptor"
 dataType                 %DataTypes;          #FIXED "Int"
>
<!ELEMENT VideoExtentEnd (#PCDATA)>
<!ATTLIST VideoExtentEnd
 superElement             NMTOKEN              #FIXED "Descriptor"
 dataType                 %DataTypes;          #FIXED "Int"
>
```

Appendix B: An Example Description Scheme for an Australian Football League Game Description Scheme (AFLGame.ddf)

```
<!-- Core.ddf included here -->
<!ENTITY % Core SYSTEM "Core.ddf">
%Core;                                                B1
            B2
<!------------------------------------------------------------------------------------
----Scheme specific entities
------------------------------------------------------------------------------------
-->
<!ENTITY AFLGameGen SYSTEM "AFLGameGen.class" NDATA
        JAVACLASS>
<!ENTITY % PlayType "(Mark | Kick | Handball | Tackle)" >
<!------------------------------------------------------------------------------------
----Element definitions                  B3
------------------------------------------------------------------------------------
->
<!ELEMENT AFLGameDescription (Game, Locator*)>
<!ATTLIST AFLGameDescription
        superElement    NMTOKEN         #FIXED "Description"
        handler         ENTITY          #FIXED "AFLGameGen"
>                                              B4
<!ELEMENT Game (Location, Date, TeamName*, Quarter*)>   <!ATTLIST Game
        superElement    NMTOKEN         #FIXED "Descriptor"
>
<!ELEMENT Location (#PCDATA)>
<!ATTLIST Location
        superElement    NMTOKEN         #FIXED "Descriptor"
>
<!ELEMENT Date (#PCDATA)>
```

B2

```
<!ATTLIST Date
        superElement    NMTOKEN         #FIXED "Descriptor"
        dataType        %DataTypes;     #FIXED "Date"
>
<!ELEMENT TeamName (#PCDATA)>
<!ATTLIST TeamName
        superElement    NMTOKEN         #FIXED "Descriptor"
>
<!ELEMENT Quarter (Play*)>
<!ATTLIST Quarter
        superElement    NMTOKEN         #FIXED "Descriptor"
>
<!ELEMENT Play (PlayerNo, PlayType, Annotator, CLink*)>
<!ATTLIST Play
        superElement    NMTOKEN         #FIXED "Descriptor"
>
<!ELEMENT PlayerNo (#PCDATA)>
<!ATTLIST PlayerNo
        superElement    NMTOKEN         #FIXED "Descriptor"
        dataType        %DataTypes;     #FIXED "Int"
>
<!ELEMENT PlayType (EMPTY)>
<!ATTLIST PlayType
        superElement    NMTOKEN         #FIXED "Descriptor"
        value           %PlayType;      #REQUIRED
>
<!ELEMENT Annotator (#PCDATA)>
<!ATTLIST Annotator
        superElement    NMTOKEN         #FIXED "Descriptor"
>
```

C)

Appendix C: An Example Description generated from the Description Scheme in Appendix B

Example Description (AFLGameEg.xml)

```
<?xml version=”1.0” standalone = “no” ?>
<!DOCTYPE AFLGameDescription SYSTEM “AFLGame.ddf” [
    <!ENTITY MatchVideo SYSTEM “MatchVideo.mpg” NDATA MPEG2>
]>
<AFLGameDescription resource = “MatchVideo”>
        <!--A description of the game is contained in this section -->
        <Game>
                <!-- First some details of the game being player -->
                <Location>Sydney Cricket Ground</Location>
                <Date>1998-08-09</Date>
                <TeamName>Sydney Swans</TeamName>
                <TeamName>West Coast Eagles</TeamName>

                <!-- Now add play information with links -->
                <Quarter id = “Q1”>
                        <Play id = “P1”>
                                <PlayerNo>23</PlayerNo>
                                <PlayType value = “Mark”/>
                                <Annotator>John Smith</Annotator>
                                <CLink linkend = “L1”/>
                        </Play>
                        <Play id = “P2”>
                                <PlayerNo>5</PlayerNo>
                                <PlayType value = “Kick”/>
                                <Annotator>Joe Bloggs</Annotator>
                                <CLink linkend = “L2”/>
                        </Play>
                </Quarter>
                <Quarter id = “Q2”> … </Quarter>
```

C2

```
        <Quarter id = "Q3"> … </Quarter>
        <Quarter id = "Q4"> … </Quarter>
    </Game>
    <!-- This section now contains the linkends for the various plays -->
    <Locator id = "L1" resource = "MatchVideo">
        <VideoExtent >
            <VideoExtentStart>0</VideoExtentStart>
            <VideoExtentEnd>10</VideoExtentEnd>
            <RectImageExtent>
                <RectImageExtentX0>50</RectImageExtentX0>
                <RectImageExtentY0>50</RectImageExtentY0>
                <RectImageExtentHeight>100</RectImageExtentHeight>
                <RectImageExtentWidth>40</RectImageExtentWidth>
            </RectImageExtent>
        </VideoExtent>
        <VideoExtent>
            <VideoExtentStart>11</VideoExtentStart>
            <VideoExtentEnd>32</VideoExtentEnd>
            <RectImageExtent>
                <RectImageExtentX0>80</RectImageExtentX0>
                <RectImageExtentY0>100</RectImageExtentY0>
                <RectImageExtentHeight>100</RectImageExtentHeight>
                <RectImageExtentWidth>40</RectImageExtentWidth>
            </RectImageExtent>
        </VideoExtent>
    </Locator>
    <Locator id = "L2" resource = "MatchVideo">
        <VideoExtent>
            <VideoExtentStart>0</VideoExtentStart>
            <VideoExtentEnd>25</VideoExtentEnd>
            <RectImageExtent>
                <RectImageExtentX0>200</RectImageExtentX0>
```

C3

```
                    <RectImageExtentY0>150</RectImageExtentY0>
                    <RectImageExtentHeight>80</RectImageExtentHeight>
                    <RectImageExtentWidth>30</RectImageExtentWidth>
                </RectImageExtent>
            </VideoExtent>
        </Locator>
</ AFLGameDescription >
```

Appendix D: Digital Video Resource Description Scheme

Description Scheme (Video.ddf)

```
<!-- Core.ddf included here -->
<!ENTITY % Core SYSTEM "Core.ddf">
%Core;

<!----------------------------------------------------------------------------------------------
----Scheme specific entities
-----------------------------------------------------------------------------------------------
-->
<!ENTITY     VideoDescGen     SYSTEM     "VideoDescGen.class"     NDATA
JAVACLASS>
<!ENTITY ShotAnalyser SYSTEM "ShotAnalyser.class" NDATA JAVACLASS>
<!ENTITY VideoPresRules SYSTEM "VideoPresentationRules.xml">

<!----------------------------------------------------------------------------------------------
---Video resource related element definitions
-----------------------------------------------------------------------------------------------
-->
                                        D1
<!ELEMENT VideoDescription (Title, Shot*, Locator*)>
<!ATTLIST VideoDescription
        superElement        NMTOKEN     #FIXED "Description"
        handler             ENTITY      #FIXED "VideoDescGen"
        ruleSets            ENTITIES    #FIXED "VideoPresRules"
>                                       D2

<!ELEMENT Title (#PCDATA)>
<!ATTLIST Title
        superElement        NMTOKEN     #FIXED "Descriptor"
>
```

D2

```
<!ELEMENT Shot (Descriptor*)>
<!ATTLIST Shot                                          D3
        superElement    NMTOKEN        #FIXED "Descriptor"
        handler         ENTITY         #FIXED "ShotAnalyser"
        keyFrame        ENTITY         #REQUIRED
        locator         IDREF          #REQUIRED
>                       D4
<!ELEMENT PeopleScene (EMPTY)>
<!ATTLIST PeopleScene      D5
        superElement    NMTOKEN        #FIXED "Descriptor"
>
<!ELEMENT CrowdScene (EMPTY)>
<!ATTLIST CrowdScene
        superElement    NMTOKEN        #FIXED "PeopleScene"
>
<!ELEMENT PortraitScene (EMPTY)>
<!ATTLIST PortraitScene
        superElement    NMTOKEN        #FIXED "PeopleScene"
>
<!ELEMENT IndoorScene (EMPTY)
<!ATTLIST IndoorScene
        superElement    NMTOKEN        #FIXED "Descriptor"
>
<!ELEMENT OutdoorScene (EMPTY)
<!ATTLIST OutdoorShot
        superElement    NMTOKEN        #FIXED "Descriptor"
>
```

Appendix E: An Example Description generated from the Video Description Scheme in Appendix D

Example Description (VideoEg1.xml)

```
<?xml version=”1.0” standalone = “no” ?>
<!DOCTYPE VideoDescription SYSTEM “Video.ddf” [
    <!ENTITY MyVideo SYSTEM “MyVideo.mpg” NDATA MPEG2>
    <!ENTITY KFrame1 SYSTEM “KFrame2.jpg” NDATA JPEG>
    <!ENTITY KFrame2 SYSTEM “KFrame2.jpg” NDATA JPEG>
    etc.
]>
<VideoDescription resource  = “MyVideo”>
        <Title>Video Clip Title</Title>
        <!-- Shots detected in the digital video resource -->
        <Shot id = “S1” keyFrame = “KFrame1” locator  = “L1”>
                <CrowdScene/>
                <OutdoorScene/>
        </Shot>
        <Shot id = “S2” keyFrame = “KFrame2” locator = “L2”>
                <PortraitScene/>
                <OutdoorScene/>
        </Shot>

        <!-- Locators in the digital video resource -->
        <Locator id = “L1” resource = “MyVideo”>
                <VideoExtent >
                        <VideoExtentStart>0</VideoExtentStart>
                        <VideoExtentEnd>20</VideoExtentEnd>
                </VideoExtent>
        </Locator>
        <Locator id = “L2”>
                <VideoExtent >
                        <VideoExtentStart>21</VideoExtentStart>
```

```
                <VideoExtentEnd>50</VideoExtentEnd>
            </VideoExtent>
        </Locator>
</VideoDescription>
```

Appendix F: Presentation Rules for the Video Description Scheme in Appendix D

Example Description (VideoPresentationRules.xml)

```
<?xml version="1.0" standalone = "no" ?>
<!DOCTYPE PresentationRules SYSTEM "Rules.dtd" [
    <!ENTITY CrowdScene SYSTEM "CrowdSceneIcon.jpg" NDATA JPEG>
    <!ENTITY PortraitScene SYSTEM "PortraitSceneIcon.jpg" NDATA JPEG>
    <!ENTITY OutdoorScene SYSTEM "OutdoorSceneIcon.jpg" NDATA JPEG>
    <!ENTITY IndoorScene SYSTEM "IndoorSceneIcon.jpg" NDATA JPEG>
]>

<PresentationRules>
        <Rule target = ElementDefn pattern = "VideoDescription">
                <Action>
                        <AddAttributeDef
                                attName = "selected"
                                attType = "CDATA"
                                attDefault = "NO"/>
                </Action>
                <Action>
                        <AddAttributeDef
                                attName = "presentationType"
                                attType = "(Index|TOC)"
                                attDefault = #FIXED "TOC"/>
                </Action>
        </Rule>
        <Rule target = ElementDefn pattern = "VideoDescription/Shot">
                <Action>
                        <AddAttributeDef
                                attName = "selected"
                                attType = "CDATA"
                                attDefault = "NO"/>
                </Action>
```

F2

```
        <Action>
            <AddAttributeDef
                attName = "presentationType"
                attType = "(Index|TOC)"
                attDefault = #FIXED "TOC"/>
        </Action>
    </Rule>
    <Rule          target          =          ElementDefn          pattern          =
"VideoDescription/Shot/CrowdScene">
        <Action>
            <AddAttributeDef
                attName = "presentationType"
                attType = "(Index|TOC)"
                attDefault = #FIXED "Index"/>
        </Action>
        <Action>
            <AddAttributeDef
                attName = "icon"
                attType = "ENTITY"
                attDefault = #FIXED "CrowdScene"/>
        </Action>
    </Rule>
    <Rule          target          =          ElementDefn          pattern          =
"VideoDescription/Shot/PortraitScene">
        <Action>
            <AddAttributeDef
                attName = "presentationType"
                attType = "(Index|TOC)"
                attDefault = #FIXED "Index"/>
        </Action>
        <Action>
            <AddAttributeDef
```

F3

```
                    attName = "icon"
                    attType = "ENTITY"
                    attDefault = #FIXED "PortraitScene"/>
          </Action>
     </Rule>
     <Rule target = ElementDefn pattern = "VideoDescription/Shot/IndoorScene">
          <Action>
               <AddAttributeDef
                    attName = "presentationType"
                    attType = "(Index|TOC)"
                    attDefault = #FIXED "Index"/>
          </Action>
          <Action>
               <AddAttributeDef
                    attName = "icon"
                    attType = "ENTITY"
                    attDefault = #FIXED "IndoorScene"/>
          </Action>
     </Rule>
     <Rule target = ElementDefn pattern =
"VideoDescription/Shot/OutdoorScene">
          <Action>
               <AddAttributeDef
                    attName = "presentationType"
                    attType = "(Index|TOC)"
                    attDefault = #FIXED "Index"/>
          </Action>
                    <Action>
               <AddAttributeDef
                    attName = "icon"
                    attType = "ENTITY"
                    attDefault = #FIXED "OutdoorScene"/>
```

F4

```
        </Action>
    </Rule>
</PresentationRules>
```

G1

Appendix G: Digital Video Library Description Scheme

Description Scheme (DigitalVideoLibrary.ddf)

```
<!-- Core.ddf included here -->
<!ENTITY % Core SYSTEM "Core.ddf">
%Core;

<!-------------------------------------------------------------------------------------------------
----Scheme specific entities
-------------------------------------------------------------------------------------------------
-->
<!ENTITY VideoLibraryGen SYSTEM "VideoLibraryGen.class"
        NDATA JAVACLASS>
<!-------------------------------------------------------------------------------------------------
---Digital Video Library related element definitions
-------------------------------------------------------------------------------------------------
-->
<!ELEMENT DigitalVideoLibraryDescription (Section* | Item*)>
<!ATTLIST DigitalVideoLibraryDescription
        superElement        NMTOKEN     #FIXED "Description"
        handler             ENTITY      #FIXED "VideoLibraryGen"
        title               CDATA       #IMPLIED
>
<!ELEMENT Section (Section* | Item*)>
<!ATTLIST Section
        superElement        NMTOKEN     #FIXED "Descriptor"
        title               CDATA       #IMPLIED
>
<!ELEMENT Item (EMPTY)>
<!ATTLIST Item
        superElement        NMTOKEN     #FIXED "Descriptor"
        description         ENTITY      #REQUIRED
>
```

Appendix H: An Example Description generated from the Digital Video Library Description Scheme in Appendix G

Example Description (VideoLibraryEg.xml)

```
<?xml version="1.0" standalone = "no" ?>
<!DOCTYPE DigitalVideoLibraryDescription SYSTEM "DigitalVideoLibrary.ddf" [
    <!ENTITY VideoEg1 SYSTEM "VideoEg1.xml">
    <!ENTITY VideoEg2 SYSTEM "VideoEg2.xml">
    <!ENTITY VideoEg3 SYSTEM "VideoEg3.xml">
    etc.
]>
<DigitalVideoLibraryDescription title = "My Personal Digital Video Library">
        <Section title = "Holiday Videos">
                <Item description = "VideoEg1"/>
                <Item description = "VideoEg2"/>
                etc.
        </Section>
        <Section title = "Birthday Videos">
                <Section title = "Mary's Birthdays">
                        <Item description = "VideoEg3/>
                        etc.
                </Section>
                <Section title = "John's Birthdays"> ... </Section>
        </Section>
</DigitalVideoLibraryDescription>
```

Appendix I: Video Presentation Description Scheme

Description Scheme (VideoPresentation.ddf)

```
<!-- Core.ddf included here -->
<!ENTITY % Core SYSTEM “Core.ddf”>
%Core;
<!--------------------------------------------------------------------------------------------------
----Scheme specific entities
----------------------------------------------------------------------------------------------------
-->
<!ENTITY VideoPresentationGen SYSTEM “VideoPresentation.class”
        NDATA JAVACLASS>
<!ENTITY VideoPresentationRules SYSTEM “VideoPresentionRules.xml”>
<!--------------------------------------------------------------------------------------------------
---Video Presentation related element definitions
----------------------------------------------------------------------------------------------------
-->
<!ELEMENT VideoPresentationDescription (VideoDescriptionReference*)>
<!ATTLIST VideoPresentationDescription
        superElement          NMTOKEN    #FIXED “Description”
        handler               ENTITY     #FIXED “VideoPresentationGen”
        title                 CDATA      #IMPLIED
        ruleSets              ENTITIES   #FIXED “VideoPresentationRules”
        userPresentationRules ENTITY     #IMPLIED
> I1                          I2
<!ELEMENT VideoDescriptionReference (ShotReference*)
<!ATTLIST VideoDescriptionReference
        superElement          NMTOKEN    #FIXED “Descriptor”
        videoDescription      ENTITY     #REQUIRED
>
<!ELEMENT ShotReference (EMPTY)
<!ATTLIST ShotReference
        superElement          NMTOKEN    #FIXED “Descriptor”
```

```
        shotIDRef           IDREF          #REQUIRED
>
```

J1

Appendix J: An Example Description generated from the Video Presentation Description Scheme in Appendix I

Example Description (VideoPresentationEg.xml)

```
<?xml version="1.0" standalone = "no" ?>
<!DOCTYPE VideoPresentationDescription SYSTEM "VideoPresentation.ddf" [
        <!ENTITY UserPresentationRules SYSTEM "UserPresentationRules.xml">
        <!ENTITY VideoEg1 SYSTEM "VideoEg1.xml">
        <!ENTITY VideoEg2 SYSTEM "VideoEg2.xml">
        etc.
]>
<VideoPresentationDescription userPresentationRules = "UserPresentationRules">
        <VideoDescriptionReference videoDescription = "VideoEg1">
              <ShotReference shotIDRef = "S1"/>    J2              J1
              <ShotReference shotIDRef = "S2"/>
        </ VideoDescriptionReference >

        <VideoDescriptionReference description = "VideoEg2">
              <ShotReference shotIDRef = "S3"/>
              <ShotReference shotIDRef = "S4"/>
        </ VideoDescriptionReference >

        etc.
</VideoPresentationDescription>
```

W3C

REC-xml-19980210

Extensible Markup Language (XML) 1.0

W3C Recommendation 10-February-1998

This version:
http://www.w3.org/TR/1998/REC-xml-19980210
http://www.w3.org/TR/1998/REC-xml-19980210.xml
http://www.w3.org/TR/1998/REC-xml-19980210.html
http://www.w3.org/TR/1998/REC-xml-19980210.pdf
http://www.w3.org/TR/1998/REC-xml-19980210.ps
Latest version:
http://www.w3.org/TR/REC-xml
Previous version:
http://www.w3.org/TR/PR-xml-971208

Editors:
Tim Bray (Textuality and Netscape) <tbray@textuality.com>
Jean Paoli (Microsoft) <jeanpa@microsoft.com>
C. M. Sperberg-McQueen (University of Illinois at Chicago) <cmsmcq@uic.edu>

Abstract

The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

Status of this document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.
This document specifies a syntax created by subsetting an existing, widely used international text processing standard (Standard Generalized Markup Language, ISO 8879:1986(E) as amended and corrected) for use on the World Wide Web. It is a product of the W3C XML Activity, details of which can be found at http://www.w3.org/XML. A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.
This specification uses the term URI, which is defined by [Berners-Lee et al.], a work in progress expected to update [IETF RFC1738] and [IETF RFC1808].
The list of known errors in this specification is available at http://www.w3.org/XML/xml-19980210-errata.
Please report errors in this document to xml-editor@w3.org.

Extensible Markup Language (XML) 1.0

Table of Contents

Appendices

---

1. Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an **XML processor** is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the **application**. This specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C. The membership of the XML Working Group is given in an appendix. Dan Connolly served as the WG's contact with the W3C.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

This specification, together with associated standards (Unicode and ISO/IEC 10646 for characters, Internet RFC 1766 for language identification tags, ISO 639 for language name codes, and ISO 3166 for country name codes), provides all the information necessary to understand XML Version 1.0 and construct computer programs to process it.

This version of the XML specification may be distributed freely, as long as all text and legal notices

remain intact.

1.2 Terminology

The terminology used to describe XML documents is defined in the body of this specification. The terms defined in the following list are used in building those definitions and in describing the actions of an XML processor:

may
: Conforming documents and XML processors are permitted to but need not behave as described.

must
: Conforming documents and XML processors are required to behave as described; otherwise they are in error.

error
: A violation of the rules of this specification; results are undefined. Conforming software may detect and report an error and may recover from it.

fatal error
: An error which a conforming XML processor must detect and report to the application. After encountering a fatal error, the processor may continue processing the data to search for further errors and may report such errors to the application. In order to support correction of errors, the processor may make unprocessed data from the document (with intermingled character data and markup) available to the application. Once a fatal error is detected, however, the processor must not continue normal processing (i.e., it must not continue to pass character data and information about the document's logical structure to the application in the normal way).

at user option
: Conforming software may or must (depending on the modal verb in the sentence) behave as described; if it does, it must provide users a means to enable or disable the behavior described.

validity constraint
: A rule which applies to all valid XML documents. Violations of validity constraints are errors; they must, at user option, be reported by validating XML processors.

well-formedness constraint
: A rule which applies to all well-formed XML documents. Violations of well-formedness constraints are fatal errors.

match
: (Of strings or names:) Two strings or names being compared must be identical. Characters with multiple possible representations in ISO/IEC 10646 (e.g. characters with both precomposed and base+diacritic forms) match only if they have the same representation in both strings. At user option, processors may normalize such characters to some canonical form. No case folding is performed. (Of strings and rules in the grammar:) A string matches a grammatical production if it belongs to the language generated by that production. (Of content and content models:) An element matches its declaration when it conforms in the fashion described in the constraint "Element Valid".

for compatibility
: A feature of XML included solely to ensure that XML remains compatible with SGML.

for interoperability
: A non-binding recommendation included to increase the chances that XML documents can be processed by the existing installed base of SGML processors which predate the WebSGML Adaptations Annex to ISO 8879.

2. Documents

A data object is an **XML document** if it is well-formed, as defined in this specification. A

well-formed XML document may in addition be valid if it meets certain further constraints. Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The logical and physical structures must nest properly, as described in "4.3.2 Well-Formed Parsed Entities".

2.1 Well-Formed XML Documents

A textual object is a well-formed XML document if:

1. Taken as a whole, it matches the production labeled document.
2. It meets all the well-formedness constraints given in this specification.
3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed.

Document

```
[1] document ::= prolog element Misc*
```

Matching the document production implies that:

1. It contains one or more elements.
2. There is exactly one element, called the **root**, or document element, no part of which appears in the content of any other element. For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

As a consequence of this, for each non-root element `C` in the document, there is one other element `P` in the document such that `C` is in the content of `P`, but is not in the content of any other element that is in the content of `P`. `P` is referred to as the **parent** of `C`, and `C` as a **child** of `P`.

2.2 Characters

A parsed entity contains **text**, a sequence of characters, which may represent markup or character data. A **character** is an atomic unit of text as specified by ISO/IEC 10646 [ISO/IEC 10646]. Legal characters are tab, carriage return, line feed, and the legal graphic characters of Unicode and ISO/IEC 10646. The use of "compatibility characters", as defined in section 6.8 of [Unicode], is discouraged.

Character Range

```
[2] Char ::= #x9 | #xA | #xD | [#x20-#xD7FF]      /* any Unicode character,
           | [#xE000-#xFFFD]                          excluding the surrogate
           | [#x10000-#x10FFFF]                       blocks, FFFE, and FFFF. */
```

The mechanism for encoding character code points into bit patterns may vary from entity to entity. All XML processors must accept the UTF-8 and UTF-16 encodings of 10646; the mechanisms for signaling which of the two is in use, or for bringing other encodings into play, are discussed later, in "4.3.3 Character Encoding in Entities".

2.3 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.

S (white space) consists of one or more space (#x20) characters, carriage returns, line feeds, or tabs.

White Space

```
[3] S ::= (#x20 | #x9 | #xD | #xA)+
```

Characters are classified for convenience as letters, digits, or other characters. Letters consist of an alphabetic or syllabic base character possibly followed by one or more combining characters, or of an ideographic character. Full definitions of the specific characters in each class are given in "B. Character Classes".

A **Name** is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters. Names beginning with the string "xml", or any string which would match (('X'|'x') ('M'|'m') ('L'|'l')), are reserved for standardization in this or future versions of this specification.

Note: The colon character within XML names is reserved for experimentation with name spaces. Its meaning is expected to be standardized at some future point, at which point those documents using the colon for experimental purposes may need to be updated. (There is no guarantee that any name-space mechanism adopted for XML will in fact use the colon as a name-space delimiter.) In practice, this means that authors should not use the colon in XML names except as part of name-space experiments, but that XML processors should accept the colon as a name character.

An Nmtoken (name token) is any mixture of name characters.

Names and Tokens

```
[4]  NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar
                  | Extender
[5]  Name     ::= (Letter | '_' | ':') (NameChar)*
[6]  Names    ::= Name (S Name)*
[7]  Nmtoken  ::= (NameChar)+
[8]  Nmtokens ::= Nmtoken (S Nmtoken)*
```

Literal data is any quoted string not containing the quotation mark used as a delimiter for that string. Literals are used for specifying the content of internal entities (EntityValue), the values of attributes (AttValue), and external identifiers (SystemLiteral). Note that a SystemLiteral can be parsed without scanning for markup.

Literals

```
[9]  EntityValue   ::= '"' ([^%&"] | PEReference | Reference)* '"'
                       | "'" ([^%&'] | PEReference | Reference)* "'"
[10] AttValue      ::= '"' ([^<&"] | Reference)* '"'
                       | "'" ([^<&'] | Reference)* "'"
[11] SystemLiteral ::= ('"' [^"]* '"') | ("'" [^']* "'")
[12] PubidLiteral  ::= '"' PubidChar* '"' | "'" (PubidChar - "'")* "'"
[13] PubidChar     ::= #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]
```

2.4 Character Data and Markup

Text consists of intermingled character data and markup. **Markup** takes the form of start-tags, end-tags, empty-element tags, entity references, character references, comments, CDATA section delimiters, document type declarations, and processing instructions.

All text that is not markup constitutes the **character data** of the document.

The ampersand character (&) and the left angle bracket (<) may appear in their literal form *only* when used as markup delimiters, or within a comment, a processing instruction, or a CDATA section. They are also legal within the literal entity value of an internal entity declaration; see "4.3.2 Well-Formed Parsed Entities". If they are needed elsewhere, they must be escaped using either numeric character references or the strings "&" and "<" respectively. The right angle bracket (>) may be represented using the string ">", and must, for compatibility, be escaped using ">" or a character reference when it appears in the string "]]>" in content, when that string is not marking the end of a CDATA section.

In the content of elements, character data is any string of characters which does not contain the

start-delimiter of any markup. In a CDATA section, character data is any string of characters not including the CDATA-section-close delimiter, "]]>".
To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') may be represented as "'", and the double-quote character (") as """.

Character Data

```
[14] CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
```

2.5 Comments

Comments may appear anywhere in a document outside other markup; in addition, they may appear within the document type declaration at places allowed by the grammar. They are not part of the document's character data; an XML processor may, but need not, make it possible for an application to retrieve the text of comments. For compatibility, the string "--" (double-hyphen) must not occur within comments.

Comments

```
[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

An example of a comment:

```
<!-- declarations for <head> & <body> -->
```

2.6 Processing Instructions

Processing instructions (PIs) allow documents to contain instructions for applications.

Processing Instructions

```
[16]       PI ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

PIs are not part of the document's character data, but must be passed through to the application. The PI begins with a target (PITarget) used to identify the application to which the instruction is directed. The target names "XML", "xml", and so on are reserved for standardization in this or future versions of this specification. The XML Notation mechanism may be used for formal declaration of PI targets.

2.7 CDATA Sections

CDATA sections may occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup. CDATA sections begin with the string "<![CDATA[" and end with the string "]]>":

CDATA Sections

```
[18]  CDSect ::= CDStart CData CDEnd
[19] CDStart ::= '<![CDATA['
[20]   CData ::= (Char* - (Char* ']]>' Char*))
[21]   CDEnd ::= ']]>'
```

Within a CDATA section, only the CDEnd string is recognized as markup, so that left angle brackets and ampersands may occur in their literal form; they need not (and cannot) be escaped using "<" and "&". CDATA sections cannot nest.
An example of a CDATA section, in which "<greeting>" and "</greeting>" are recognized as character data, not markup:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

2.8 Prolog and Document Type Declaration

XML documents may, and should, begin with an **XML declaration** which specifies the version of XML being used. For example, the following is a complete XML document, well-formed but not valid:

```
<?xml version="1.0"?>
<greeting>Hello, world!</greeting>
```

and so is this:

```
<greeting>Hello, world!</greeting>
```

The version number " 1.0 " should be used to indicate conformance to this version of this specification; it is an error for a document to use the value "1.0" if it does not conform to this version of this specification. It is the intent of the XML working group to give later versions of this specification numbers other than "1.0", but this intent does not indicate a commitment to produce any future versions of XML, nor if any are produced, to use any particular numbering scheme. Since future versions are not ruled out, this construct is provided as a means to allow the possibility of automatic version recognition, should it become necessary. Processors may signal an error if they receive documents labeled with versions they do not support.

The function of the markup in an XML document is to describe its storage and logical structure and to associate attribute-value pairs with its logical structures. XML provides a mechanism, the document type declaration, to define constraints on the logical structure and to support the use of predefined storage units. An XML document is **valid** if it has an associated document type declaration and if the document complies with the constraints expressed in it.

The document type declaration must appear before the first element in the document.

Prolog

```
[22]      prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
[23]     XMLDecl ::= '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24] VersionInfo ::= S 'version' Eq (' VersionNum ' | " VersionNum ")
[25]          Eq ::= S? '=' S?
[26]  VersionNum ::= ([a-zA-Z0-9_.:] | '-')+
[27]        Misc ::= Comment | PI | S
```

The XML **document type declaration** contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or **DTD**. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. The DTD for a document consists of both subsets taken together.

A **markup declaration** is an element type declaration, an attribute-list declaration, an entity declaration, or a notation declaration. These declarations may be contained in whole or in part within parameter entities, as described in the well-formedness and validity constraints below. For fuller information, see "4. Physical Structures".

Document Type Definition

```
[28] doctypedecl ::= '<!DOCTYPE' S Name (S           [ VC: Root Element Type ]
                     ExternalID)? S? ('[' (markupdecl
                     | PEReference | S)* ']' S?)? '>'
[29]  markupdecl ::= elementdecl | AttlistDecl       [ VC: Proper
                     | EntityDecl | NotationDecl | PI  Declaration/PE Nesting
                     | Comment                         ]
                                                     [ WFC: PEs in Internal
                                                     Subset ]
```

The markup declarations may be made up in whole or in part of the replacement text of parameter entities. The productions later in this specification for individual nonterminals (elementdecl, AttlistDecl, and so on) describe the declarations *after* all the parameter entities have been included .

Validity Constraint: Root Element Type

The Name in the document type declaration must match the element type ofthe root element.

Validity Constraint: Proper Declaration/PE Nesting

Parameter-entity replacement text must be properly nested with markup declarations. That is to say, if either the first character or the last character of a markup declaration (markupdecl above) is contained in the replacement text for a parameter-entity reference, both must be contained in the same replacement text.

Well-Formedness Constraint: PEs in Internal Subset

In the internal DTD subset, parameter-entity references can occur only where markup declarations can occur, not within markup declarations. (This does not apply to references that occur in external parameter entities or to the external subset.)

Like the internal subset, the external subset and any external parameter entities referred to in the DTD must consist of a series of complete markup declarations of the types allowed by the non-terminal symbol markupdecl, interspersed with white space or parameter-entity references . However, portions of the contents of the external subset or of external parameter entities may conditionally be ignored by using the conditional section construct; this is not allowed in the internal subset.

External Subset

```
[30]  extSubset ::= TextDecl? extSubsetDecl
[31]  extSubsetDecl ::= ( markupdecl | conditionalSect | PEReference | S )*
```

The external subset and external parameter entities also differ from the internal subset in that in them, parameter-entity references are permitted *within* markup declarations, not only *between* markup declarations.

An example of an XML document with a document typedeclaration:

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

The system identifier "hello.dtd" gives the URI of a DTD for thedocument.

The declarations can also be given locally, as in this example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

If both the external and internal subsets are used, the internal subset is considered to occur before the external subset. This has the effect that entity and attribute-list declarations in the internal subset take precedence over those in the external subset.

2.9 Standalone Document Declaration

Markup declarations can affect the content of the document, as passed from an XML processor to an application; examples are attribute defaults and entity declarations. The standalone document declaration, which may appear as a component of the XML declaration, signals whether or not there are such declarations which appear external to the document entity.

Standalone Document Declaration

```
[32]  SDDecl ::= S 'standalone' Eq (("'" ('yes' | 'no') "'") | ('"' ('yes' | 'no') '"'))   [ VC: Standalone Document Declaration ]
```

In a standalone document declaration, the value "yes" indicates that there are no markup declarations external to the document entity (either in the DTD external subset, or in an external parameter entity referenced from the internal subset) which affect the information passed from the XML processor to the application. The value "no" indicates that there are or may be such external markup declarations. Note that the standalone document declaration only denotes the presence of external *declarations*; the presence, in a document, of references to external *entities*, when those entities are internally declared, does not change its standalone status.
If there are no external markup declarations, the standalone document declaration has no meaning. If there are external markup declarations but there is no standalone document declaration, the value "no" is assumed.
Any XML document for which standalone="no" holds can be converted algorithmically to a standalone document, which may be desirable for some network delivery applications.
Validity Constraint: Standalone Document Declaration
The standalone document declaration must have the value "no" if any external markup declarations contain declarations of:

- attributes with default values, if elements to which these attributes apply appear in th document without specifications of values for these attributes, or
- entities (other than amp, lt, gt, apos, quot), if references to those entities appear in the document, or
- attributes with values subject to normalization, where the attribute appears in the document with a value which will change as a result of normalization, or
- element types with element content, if white space occurs directly within any instance of those types.

An example XML declaration with a standalone document declaration:

```
<?xml version="1.0" standalone='yes'?>
```

2.10 White Space Handling

In editing XML documents, it is often convenient to use "white space" (spaces, tabs, and blank lines, denoted by the nonterminal S in this specification) to set apart the markup for greater readability. Such white space is typically not intended for inclusion in the delivered version of the document. On the other hand, "significant" white space that should be preserved in the delivered version is common, for example in poetry and source code.
An XML processor must always pass all characters in a document that are not markup through to the application. A validating XML processor must also inform the application which of these characters constitute white space appearing in element content.
A special attribute named xml:space may be attached to an element to signal an intention that in that element, white space should be preserved by applications. In valid documents, this attribute, like any other, must be declared if it is used. When declared, it must be given as an enumerated type whose only possible values are "default" and "preserve". For example:

```
    <!ATTLIST poem   xml:space (default|preserve) 'preserve'>
```

The value "default" signals that applications' default white-space processing modes are acceptable for this element; the value "preserve" indicates the intent that applications preserve all the white space. This declared intent is considered to apply to all elements within the content of the element where it is specified, unless overriden with another instance of the xml:space attribute.
The root element of any document is considered to have signaled no intentions as regards application space handling, unless it provides a value for this attribute or the attribute is declared with a default value.

2.11 End-of-Line Handling

XML parsed entities are often stored in computer files which, for editing convenience, are organized into lines. These lines are typically separated by some combination of the characters carriage-return

(#xD) and line-feed (#xA).

To simplify the tasks of applications, wherever an external parsed entity or the literal entity value of an internal parsed entity contains either the literal two-character sequence "#xD#xA" or a standalone literal #xD, an XML processor must pass to the application the single character #xA. (This behavior can conveniently be produced by normalizing all line breaks to #xA on input, before parsing.)

2.12 Language Identification

In document processing, it is often useful to identify the natural or formal language in which the content is written. A special attribute named xml:lang may be inserted in documents to specify the language used in the contents and attribute values of any element in an XML document. In valid documents, this attribute, like any other, must be declared if it is used. The values of the attribute are language identifiers as defined by [IETF RFC 1766], "Tags for the Identification of Languages":

Language Identification

```
[33] LanguageID ::= Langcode ('-' Subcode)*
[34]   Langcode ::= ISO639Code | IanaCode | UserCode
[35] ISO639Code ::= ([a-z] | [A-Z]) ([a-z] | [A-Z])
[36]   IanaCode ::= ('i' | 'I') '-' ([a-z] | [A-Z])+
[37]   UserCode ::= ('x' | 'X') '-' ([a-z] | [A-Z])+
[38]    Subcode ::= ([a-z] | [A-Z])+
```

The Langcode may be any of the following:

- a two-letter language code as defined by [ISO 639], "Codes for the representation of names of languages"
- a language identifier registered with the Internet Assigned Numbers Authority [IANA]; these begin with the prefix "i-" (or "I-")
- a language identifier assigned by the user, or agreed on between parties in private use; these must begin with the prefix "x-" or "X-" in order to ensure that they do not conflict with names later standardized or registered with IANA

There may be any number of Subcode segments; if the first subcode segment exists and the Subcode consists of two letters, then it must be a country code from [ISO 3166], "Codes for the representation of names of countries." If the first subcode consists of more than two letters, it must be a subcode for the language in question registered with IANA, unless the Langcode begins with the prefix "x-" or "X-".

It is customary to give the language code in lower case, and the country code (if any) in upper case. Note that these values, unlike other names in XML documents, are case insensitive. For example:

```
<p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
  <l>Habe nun, ach! Philosophie,</l>
  <l>Juristerei, und Medizin</l>
  <l>und leider auch Theologie</l>
  <l>durchaus studiert mit heißem Bemüh'n.</l>
</sp>
```

The intent declared with xml:lang is considered to apply to all attributes and content of the element where it is specified, unless overridden with an instance of xml:lang on another element within that content.

A simple declaration for xml:lang might take the form

```
xml:lang NMTOKEN #IMPLIED
```

but specific default values may also be given, if appropriate. In a collection of French poems for English students, with glosses and notes in English, the xml:lang attribute might be declared this way:

```
<!ATTLIST poem   xml:lang NMTOKEN 'fr'>
<!ATTLIST gloss  xml:lang NMTOKEN 'en'>
<!ATTLIST note   xml:lang NMTOKEN 'en'>
```

3. Logical Structures

Each XML document contains one or more **elements**, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, sometimes called its "generic identifier" (GI), and may have a set of attribute specifications. Each attribute specification has a name and a value.

Element

```
[39] element ::= EmptyElemTag
                 | STag content ETag [ WFC: Element Type Match ]
                                     [ VC: Element Valid ]
```

This specification does not constrain the semantics, use, or (beyond syntax) names of the element types and attributes, except that names beginning with a match to (('X'|'x')('M'|'m')('L'|'l'))are reserved for standardization in this or future versions of this specification.

Well-Formedness Constraint: Element Type Match

The Name in an element's end-tag must match the element type in the start-tag.

Validity Constraint: Element Valid

An element is valid if there is a declaration matching elementdecl where the Name matches the element type, and one of the following holds:

1. The declaration matches EMPTY and the element has no content.
2. The declaration matches children and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space (characters matching the nonterminal S) between each pair of child elements.
3. The declaration matches Mixed and the content consists of character data and child elements whose types match names in the content model.
4. The declaration matches ANY, and the types of any child elements have been declared.

3.1 Start-Tags, End-Tags, and Empty-Element Tags

The beginning of every non-empty XML element is marked by a **start-tag**.

Start-tag

```
[40] STag      ::= '<' Name (S Attribute)* S? '>'  [ WFC: Unique Att Spec ]
[41] Attribute ::= Name Eq AttValue                [ VC: Attribute Value Type ]
                                                   [ WFC: No External Entity References ]
                                                   [ WFC: No < in Attribute Values ]
```

The Name in the start- and end-tags gives the element's **type**. The Name-AttValue pairs are referred to as the **attribute specifications** of the element, with the Name in each pair referred to as the **attribute name** and the content of the AttValue (the text between the ' or " delimiters) as the **attribute value**.

Well-Formedness Constraint: Unique Att Spec

No attribute name may appear more than once in the same start-tag or empty-element tag.

Validity Constraint: Attribute Value Type

The attribute must have been declared; the value must be of the type declared for it. (For attribute types, see "3.3 Attribute-List Declarations".)

Well-Formedness Constraint: No External Entity References

Attribute values cannot contain direct or indirect entity references to external entities.

Well-Formedness Constraint: No < in Attribute Values
The replacement text of any entity referred to directly or indirectly in an attribute value (other than "<") must not contain a <.
An example of a start-tag:

```
<termdef id="dt-dog" term="dog">
```

The end of every element that begins with a start-tag must be marked by an **end-tag** containing a name that echoes the element's type as given in the start-tag:

End-tag

```
[42] ETag ::= '</' Name S? '>'
```

An example of an end-tag:

```
</termdef>
```

The text between the start-tag and end-tag is called the element's **content**:

Content of Elements

```
[43] content ::= (element | CharData | Reference | CDSect | PI | Comment)*
```

If an element is **empty**, it must be represented either by a start-tag immediately followed by an end-tag or by an empty-element tag. An **empty-element tag** takes a special form:

Tags for Empty Elements

```
[44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>' [ WFC: Unique Att Spec ]
```

Empty-element tags may be used for any element which has no content, whether or not it is declared using the keyword EMPTY. For interoperability, the empty-element tag must be used, and can only be used, for elements which are declared EMPTY.
Examples of empty elements:

```
<IMG align="left"
 src="http://www.w3.org/Icons/WWW/w3c_home" />
<br></br>
<br/>
```

3.2 Element Type Declarations

The element structure of an XML document may, for validation purposes, be constrained using element type and attribute-list declarations. An element type declaration constrains the element's content.
Element type declarations often constrain which element types can appear as children of the element. At user option, an XML processor may issue a warning when a declaration mentions an element type for which no declaration is provided, but this is not an error.
An **element type declaration** takes the form:

Element Type Declaration

```
[45] elementdecl ::= '<!ELEMENT' S Name S          [ VC: Unique Element Type
                     contentspec S? '>'              Declaration ]
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed
                     | children
```

where the Name gives the element type being declared.
Validity Constraint: Unique Element Type Declaration
No element type may be declared more than once.
Examples of element type declarations:

```
<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA|emph)* >
<!ELEMENT %name.para; %content.para; >
<!ELEMENT container ANY>
```

3.2.1 Element Content

An element type has **element content** when elements of that type must contain only child elements (no character data), optionally separated by white space (characters matching the nonterminal S). In this case, the constraint includes a content model, a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear. The grammar is built on content particles (cps), which consist of names, choice lists of content particles, or sequence lists of content particles:

Element-content Models

```
[47] children ::= (choice | seq) ('?' | '*'
                  | '+')?
[48]       cp ::= (Name | choice | seq) ('?' | '*'
                  | '+')?
[49]   choice ::= '(' S? cp ( S? '|' S? cp )* S?     [ VC: Proper Group/PE
                  ')'                                   Nesting ]
[50]      seq ::= '(' S? cp ( S? ',' S? cp )* S?     [ VC: Proper Group/PE
                  ')'                                   Nesting ]
```

where each Name is the type of an element which may appear as a child. Any content particle in a choice list may appear in the element content at the location where the choice list appears in the grammar; content particles occurring in a sequence list must each appear in the element content in the order given in the list. The optional character following a name or list governs whether the element or the content particles in the list may occur one or more (+), zero or more (*), or zero or one times (?). The absence of such an operator means that the element or content particle must appear exactly once. This syntax and meaning are identical to those used in the productions in this specification.

The content of an element matches a content model if and only if it is possible to trace out a path through the content model, obeying the sequence, choice, and repetition operators and matching each element in the content against an element type in the content model. For compatibility, it is an error if an element in the document can match more than one occurrence of an element type in the content model. For more information, see "E. Deterministic ContentModels".

Validity Constraint: Proper Group/PE Nesting

Parameter-entity replacement text must be properly nested with parenthetized groups. That is to say, if either of the opening or closing parentheses in a choice, seq, or Mixed construct is contained in the replacement text for a parameter entity, both must be contained in the same replacement text. For interoperability, if a parameter-entity reference appears in a choice, seq, or Mixed construct, its replacement text should not be empty, and neither the first nor last non-blank character of the replacement text should be a connector (| or ,).

Examples of element-content models:

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
<!ELEMENT dictionary-body (%div.mix; | %dict.mix;)*>
```

3.2.2 Mixed Content

An element type has **mixed content** when elements of that type may contain character data, optionally interspersed with child elements. In this case, the types of the child elements may be constrained, but not their order or their number of occurrences:

Mixed-content Declaration

```
[51] Mixed ::= '(' S? '#PCDATA' (S? '|' S? Name)*
               S? ')*'
             | '(' S? '#PCDATA' S? ')'          [ VC: Proper Group/PE
                                                  Nesting ]
                                                [ VC: No Duplicate Types ]
```

where the Names give the types of elements that may appear as children.

Validity Constraint: No Duplicate Types

The same name must not appear more than once in a single mixed-content declaration.

Examples of mixed content declarations:

```
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >
<!ELEMENT b (#PCDATA)>
```

3.3 Attribute-List Declarations

Attributes are used to associate name-value pairs with elements. Attribute specifications may appear only within start-tags and empty-element tags; thus, the productions used to recognize them appear in "3.1 Start-Tags, End-Tags, and Empty-Element Tags". Attribute-list declarations may be used:

- To define the set of attributes pertaining to a given element type.
- To establish type constraints for these attributes.
- To provide default values for attributes.

Attribute-list declarations specify the name, data type, and default value (if any) of each attribute associated with a given element type:

Attribute-list Declaration

```
[52] AttlistDecl ::= '<!ATTLIST' S Name AttDef* S? '>'
[53]      AttDef ::= S Name S AttType S DefaultDecl
```

The Name in the AttlistDecl rule is the type of an element. At user option, an XML processor may issue a warning if attributes are declared for an element type not itself declared, but this is not an error. The Name in the AttDef rule is the name of the attribute.

When more than one AttlistDecl is provided for a given element type, the contents of all those provided are merged. When more than one definition is provided for the same attribute of a given element type, the first declaration is binding and later declarations are ignored. For interoperability, writers of DTDs may choose to provide at most one attribute-list declaration for a given element type, at most one attribute definition for a given attribute name, and at least one attribute definition in each attribute-list declaration. For interoperability, an XML processor may at user option issue a warning when more than one attribute-list declaration is provided for a given element type, or more than one attribute definition is provided for a given attribute, but this is not an error.

3.3.1 Attribute Types

XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The string type may take any literal string as a value; the tokenized types have varying lexical and semantic constraints, as noted:

Attribute Types

[54]	AttType	::=	StringType | TokenizedType | EnumeratedType	
[55]	StringType	::=	'CDATA'	
[56]	TokenizedType	::=	'ID'	[VC: ID] [VC: One ID per Element Type] [VC: ID Attribute Default]
			| 'IDREF'	[VC: IDREF]
			| 'IDREFS'	[VC: IDREF]
			| 'ENTITY'	[VC: Entity Name]
			| 'ENTITIES'	[VC: Entity Name]
			| 'NMTOKEN'	[VC: Name Token]
			| 'NMTOKENS'	[VC: Name Token]

Validity Constraint: ID
Values of type ID must match the Name production. A name must not appear more than once in an XML document as a value of this type; i.e., ID values must uniquely identify the elements which bear them.
Validity Constraint: One ID per Element Type
No element type may have more than one ID attribute specified.
Validity Constraint: ID Attribute Default
An ID attribute must have a declared default of #IMPLIED or #REQUIRED.
Validity Constraint: IDREF
Values of type IDREF must match the Name production, and values of type IDREFS must match Names; each Name must match the value of an ID attribute on some element in the XML document; i.e. IDREF values must match the value of some ID attribute.
Validity Constraint: Entity Name
Values of type ENTITY must match the Name production, values of type ENTITIES must match Names; each Name must match the name of an unparsed entity declared in the DTD.
Validity Constraint: Name Token
Values of type NMTOKEN must match the Nmtoken production; values of type NMTOKENS must match Nmtokens.
Enumerated attributes can take one of a list of values provided in the declaration. There are two kinds of enumerated types:

Enumerated Attribute Types

[57]	EnumeratedType	::=	NotationType | Enumeration	
[58]	NotationType	::=	'NOTATION' S '(' S? Name (S? '|' S? Name)* S? ')'	[VC: Notation Attributes]
[59]	Enumeration	::=	'(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'	[VC: Enumeration]

A NOTATION attribute identifies a notation, declared in the DTD with associated system and/or public identifiers, to be used in interpreting the element to which the attribute is attached.
Validity Constraint: Notation Attributes
Values of this type must match one of the notation names included in the declaration; all notation names in the declaration must be declared.
Validity Constraint: Enumeration
Values of this type must match one of the Nmtoken tokens in the declaration.
For interoperability, the same Nmtoken should not occur more than once in the enumerated attribute types of a single element type.

3.3.2 Attribute Defaults

An attribute declaration provides information on whether the attribute's presence is required, and if not, how an XML processor should react if a declared attribute is absent in a document.

Attribute Defaults

```
[60] DefaultDecl ::= '#REQUIRED' | '#IMPLIED'
                   | (('#FIXED' S)? AttValue) [ VC: Required Attribute ]
                                              [ VC: Attribute Default Legal ]
                                              [ WFC: No < in Attribute Values
                                              ]
                                              [ VC: Fixed Attribute Default ]
```

In an attribute declaration, #REQUIRED means that the attribute must always be provided, #IMPLIED that no default value is provided. If the declaration is neither #REQUIRED nor #IMPLIED, then the AttValue value contains the declared **default** value; the #FIXED keyword states that the attribute must always have the default value. If a default value is declared, when an XML processor encounters an omitted attribute, it is to behave as though the attribute were present with the declared default value.

Validity Constraint: Required Attribute

If the default declaration is the keyword #REQUIRED, then the attribute must be specified for all elements of the type in the attribute-list declaration.

Validity Constraint: Attribute Default Legal

The declared default value must meet the lexical constraints of the declared attribute type.

Validity Constraint: Fixed Attribute Default

If an attribute has a default value declared with the #FIXED keyword, instances of that attribute must match the default value.

Examples of attribute-list declarations:

```
<!ATTLIST termdef
          id      ID      #REQUIRED
          name    CDATA   #IMPLIED>
<!ATTLIST list
          type    (bullets|ordered|glossary)  "ordered">
<!ATTLIST form
          method  CDATA   #FIXED "POST">
```

3.3.3 Attribute-Value Normalization

Before the value of an attribute is passed to the application or checked for validity, the XML processor must normalize it as follows:

- a character reference is processed by appending the referenced character to the attribute value
- an entity reference is processed by recursively processing the replacement text of the entity
- a whitespace character (#x20, #xD, #xA, #x9) is processed by appending #x20 to th normalized value, except that only a single #x20 is appended for a "#xD#xA" sequence that is part of an external parsed entity or the literal entity value of an internal parsed entity
- other characters are processed by appending them to the normalized value

If the declared value is not CDATA, then the XML processor must further process the normalized attribute value by discarding any leading and trailing space (#x20) characters, and by replacing sequences of space (#x20) characters by a single space (#x20) character.

All attributes for which no declaration has been read should be treated by a non-validating parser as if declared CDATA.

3.4 Conditional Sections

Conditional sections are portions of the document type declaration external subset which are included in, or excluded from, the logical structure of the DTD based on the keyword which governs them.

Conditional Section

```
[61]     conditionalSect ::= includeSect | ignoreSect
[62]         includeSect ::= '<![' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'
[63]          ignoreSect ::= '<![' S? 'IGNORE' S? '[' ignoreSectContents* ']]>'
[64] ignoreSectContents ::= Ignore ('<![' ignoreSectContents ']]>' Ignore)*
[65]              Ignore ::= Char* - (Char* ('<![' | ']]>') Char*)
```

Like the internal and external DTD subsets, a conditional section may contain one or more complete declarations, comments, processing instructions, or nested conditional sections, intermingled with white space.

If the keyword of the conditional section is INCLUDE, then the contents of the conditional section are part of the DTD. If the keyword of the conditional section is IGNORE, then the contents of the conditional section are not logically part of the DTD. Note that for reliable parsing, the contents of even ignored conditional sections must be read in order to detect nested conditional sections and ensure that the end of the outermost (ignored) conditional section is properly detected. If a conditional section with a keyword of INCLUDE occurs within a larger conditional section with a keyword of IGNORE, both the outer and the inner conditional sections are ignored.

If the keyword of the conditional section is a parameter-entity reference, the parameter entity must be replaced by its content before the processor decides whether to include or ignore the conditional section.

An example:

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >

<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

4. Physical Structures

An XML document may consist of one or many storage units. These are called **entities**; they all have **content** and are all (except for the document entity, see below, and the external DTD subset) identified by **name**. Each XML document has one entity called the document entity, which serves as the starting point for the XML processor and may contain the whole document. Entities may be either parsed or unparsed. A **parsed entity's** contents are referred to as its replacement text; this text is considered an integral part of the document.

An **unparsed entity** is a resource whose contents may or may not be text, and if text, may not be XML. Each unparsed entity has an associated notation, identified by name. Beyond a requirement that an XML processor make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.

Parsed entities are invoked by name using entity references; unparsed entities by name, given in the value of ENTITY or ENTITIES attributes.

General entities are entities for use within the document content. In this specification, general entities are sometimes referred to with the unqualified term *entity* when this leads to no ambiguity. Parameter entities are parsed entities for use within the DTD. These two types of entities use different forms of reference and are recognized in different contexts. Furthermore, they occupy different namespaces; a parameter entity and a general entity with the same name are two distinct entities.

4.1 Character and Entity References

A **character reference** refers to a specific character in the ISO/IEC 10646 character set, for example one not directly accessible from available input devices.

Character Reference

```
[66] CharRef ::= '&#' [0-9]+ ';'
               | '&#x' [0-9a-fA-F]+ ';'  [ WFC: Legal Character ]
```

Well-Formedness Constraint: Legal Character

Characters referred to using character references must match the production for Char.

If the character reference begins with "&#x", the digits and letters up to the terminating ; provide a hexadecimal representation of the character's code point in ISO/IEC 10646. If it begins just with "&#", the digits up to the terminating ; provide a decimal representation of the character's code point.

An **entity reference** refers to the content of a named entity. References to parsed general entities use ampersand (&) and semicolon (;) as delimiters. **Parameter-entity references** use percent-sign (%) and semicolon (;) as delimiters.

Entity Reference

```
[67] Reference   ::= EntityRef | CharRef
[68] EntityRef   ::= '&' Name ';'   [ WFC: Entity Declared ]
                                    [ VC: Entity Declared ]
                                    [ WFC: Parsed Entity ]
                                    [ WFC: No Recursion ]
[69] PEReference ::= '%' Name ';'   [ VC: Entity Declared ]
                                    [ WFC: No Recursion ]
                                    [ WFC: In DTD ]
```

Well-Formedness Constraint: Entity Declared

In a document without any DTD, a document with only an internal DTD subset which contains no parameter entity references, or a document with "standalone='yes'", the Name given in the entity reference must match that in an entity declaration, except that well-formed documents need not declare any of the following entities: amp, lt, gt, apos, quot. The declaration of a parameter entity must precede any reference to it. Similarly, the declaration of a general entity must precede any reference to it which appears in a default value in an attribute-list declaration. Note that if entities are declared in the external subset or in external parameter entities, a non-validating processor is not obligated to read and process their declarations; for such documents, the rule that an entity must be declared is a well-formedness constraint only if standalone='yes'.

Validity Constraint: Entity Declared

In a document with an external subset or external parameter entities with "standalone='no'", the Name given in the entity reference must match that in an entity declaration. For interoperability, valid documents should declare the entities amp, lt, gt, apos, quot, in the form specified in "4.6 Predefined Entities". The declaration of a parameter entity must precede any reference to it. Similarly, the declaration of a general entity must precede any reference to it which appears in a default value in an attribute-list declaration.

Well-Formedness Constraint: Parsed Entity

An entity reference must not contain the name of an unparsed entity. Unparsed entities may be referred to only in attribute values declared to be of type ENTITY or ENTITIES.

Well-Formedness Constraint: No Recursion

A parsed entity must not contain a recursive reference to itself, either directly or indirectly.

Well-Formedness Constraint: In DTD

Parameter-entity references may only appear in the DTD.

Examples of character and entity references:

```
Type <key>less-than</key> (&#x3C;) to save options.
This document was prepared on &docdate; and
is classified &security-level;.
```

Example of a parameter-entity reference:

```
<!-- declare the parameter entity "ISOLat2"... -->
<!ENTITY % ISOLat2
        SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ... now reference it. -->
%ISOLat2;
```

4.2 Entity Declarations

Entities are declared thus:

Entity Declaration

[70]	EntityDecl	::=	GEDecl | PEDecl	
[71]	GEDecl	::=	'<!ENTITY' S Name S EntityDef S? '>'	
[72]	PEDecl	::=	'<!ENTITY' S '%' S Name S PEDef S? '>'	
[73]	EntityDef	::=	EntityValue | (ExternalID NDataDecl?)	
[74]	PEDef	::=	EntityValue | ExternalID	

The Name identifies the entity in an entity reference or, in the case of an unparsed entity, in the value of an ENTITY or ENTITIES attribute. If the same entity is declared more than once, the first declaration encountered is binding; at user option, an XML processor may issue a warning if entities are declared multiple times.

4.2.1 Internal Entities

If the entity definition is an EntityValue, the defined entity is called an **internal entity**. There is no separate physical storage object, and the content of the entity is given in the declaration. Note that some processing of entity and character references in the literal entity value may be required to produce the correct replacement text: see "4.5 Construction of Internal Entity Replacement Text". An internal entity is a parsed entity.

Example of an internal entity declaration:

```
<!ENTITY Pub-Status "This is a pre-release of the
 specification.">
```

4.2.2 External Entities

If the entity is not internal, it is an **external entity**, declared as follows:

External Entity Declaration

[75]	ExternalID	::=	'SYSTEM' S SystemLiteral	
			| 'PUBLIC' S PubidLiteral S SystemLiteral	
[76]	NDataDecl	::=	S 'NDATA' S Name	[VC: Notation Declared]

If the NDataDecl is present, this is a general unparsed entity; otherwise it is a parsed entity.
Validity Constraint: Notation Declared
The Name must match the declared name of a notation.
The SystemLiteral is called the entity's **system identifier**. It is a URI, which may be used to retrieve the entity. Note that the hash mark (#) and fragment identifier frequently used with URIs are not, formally, part of the URI itself; an XML processor may signal an error if a fragment identifier is given as part of a system identifier. Unless otherwise provided by information outside the scope of this specification (e.g. a special XML element type defined by a particular DTD, or a processing instruction defined by a particular application specification), relative URIs are relative to the location of the resource within which the entity declaration occurs. A URI might thus be relative to

the document entity, to the entity containing the external DTD subset, or to some other external parameter entity.

An XML processor should handle a non-ASCII character in a URI by representing the character in UTF-8 as one or more bytes, and then escaping these bytes with the URI escaping mechanism (i.e., by converting each byte to %HH, where HH is the hexadecimal notation of the byte value).

In addition to a system identifier, an external identifier may include a **public identifier**. An XML processor attempting to retrieve the entity's content may use the public identifier to try to generate an alternative URI. If the processor is unable to do so, it must use the URI specified in the system literal. Before a match is attempted, all strings of white space in the public identifier must be normalized to single space characters (#x20), and leading and trailing white space must be removed.

Examples of external entity declarations:

```
<!ENTITY open-hatch
         SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
         SYSTEM "../grafix/OpenHatch.gif"
         NDATA gif >
```

4.3 Parsed Entities

4.3.1 The Text Declaration

External parsed entities may each begin with a **text declaration**.

Text Declaration

```
[77] TextDecl ::= '<?xml' VersionInfo? EncodingDecl S? '?>'
```

The text declaration must be provided literally, not by reference to a parsed entity. No text declaration may appear at any position other than the beginning of an external parsed entity.

4.3.2 Well-Formed Parsed Entities

The document entity is well-formed if it matches the production labeled document. An external general parsed entity is well-formed if it matches the production labeled extParsedEnt. An external parameter entity is well-formed if it matches the production labeled extPE.

Well-Formed External Parsed Entity

```
[78] extParsedEnt ::= TextDecl? content
[79]        extPE ::= TextDecl? extSubsetDecl
```

An internal general parsed entity is well-formed if its replacement text matches the production labeled content. All internal parameter entities are well-formed by definition.

A consequence of well-formedness in entities is that the logical and physical structures in an XML document are properly nested; no start-tag, end-tag, empty-element tag, element, comment, processing instruction, character reference, or entity reference can begin in one entity and end in another.

4.3.3 Character Encoding in Entities

Each external parsed entity in an XML document may use a different encoding for its characters. All XML processors must be able to read entities in either UTF-8 or UTF-16.

Entities encoded in UTF-16 must begin with the Byte Order Mark described by ISO/IEC 10646 Annex E and Unicode Appendix B (the ZERO WIDTH NO-BREAK SPACE character, #xFEFF). This is an encoding signature, not part of either the markup or the character data of the XML

document. XML processors must be able to use this character to differentiate between UTF-8 and UTF-16 encoded documents.

Although an XML processor is required to read only entities in the UTF-8 and UTF-16 encodings, it is recognized that other encodings are used around the world, and it may be desired for XML processors to read entities that use them. Parsed entities which are stored in an encoding other than UTF-8 or UTF-16 must begin with a text declaration containing an encoding declaration:

Encoding Declaration

```
[80] EncodingDecl ::= S 'encoding' Eq ('"' EncName
                      '"' | "'" EncName "'" )
[81]      EncName ::= [A-Za-z] ([A-Za-z0-9._]          /* Encoding name contains
                      | '-')*                             only Latin characters
                                                          */
```

In the document entity, the encoding declaration is part of the XML declaration. The EncName is the name of the encoding used.

In an encoding declaration, the values "UTF-8", "UTF-16", "ISO-10646-UCS-2", and "ISO-10646-UCS-4" should be used for the various encodings and transformations of Unicode / ISO/IEC 10646, the values "ISO-8859-1", "ISO-8859-2", ... "ISO-8859-9" should be used for the parts of ISO 8859, and the values "ISO-2022-JP", "Shift_JIS", and "EUC-JP" should be used for the various encoded forms of JIS X-0208-1997. XML processors may recognize other encodings; it is recommended that character encodings registered (as *charsets*) with the Internet Assigned Numbers Authority [IANA], other than those just listed, should be referred to using their registered names. Note that these registered names are defined to be case-insensitive, so processors wishing to match against them should do so in a case-insensitive way.

In the absence of information provided by an external transport protocol (e.g. HTTP or MIME), it is an error for an entity including an encoding declaration to be presented to the XML processor in an encoding other than that named in the declaration, for an encoding declaration to occur other than at the beginning of an external entity, or for an entity which begins with neither a Byte Order Mark nor an encoding declaration to use an encoding other than UTF-8. Note that since ASCII is a subset of UTF-8, ordinary ASCII entities do not strictly need an encoding declaration.

It is a fatal error when an XML processor encounters an entity with an encoding that it is unable to process.

Examples of encoding declarations:

```
<?xml encoding='UTF-8'?>
<?xml encoding='EUC-JP'?>
```

4.4 XML Processor Treatmentof Entities and References

The table below summarizes the contexts in which character references, entity references, and invocations of unparsed entities might appear and the required behavior of an XML processor in each case. The labels in the leftmost column describe the recognition context:

Reference in Content
: as a reference anywhere after the start-tag and before the end-tag of an element; corresponds to the nonterminal content.

Reference in Attribute Value
: as a reference within either the value of an attribute in a start-tag, or a default value in an attribute declaration; corresponds to the nonterminal AttValue.

Occurs as Attribute Value
: as a Name, not a reference, appearing either as the value of an attribute which has been declared as type ENTITY, or as one of the space-separated tokens in the value of an attribute which has been declared as type ENTITIES.

Reference in Entity Value
: as a reference within a parameter or internal entity's literal entity value in the entity's

declaration; corresponds to the nonterminal EntityValue.

Reference in DTD

as a reference within either the internal or external subsets of the DTD, but outside of an EntityValue or AttValue.

	Entity Type			
	Parameter	Internal General	External Parsed General	Unparsed
Reference in Content	Not recognized	Included	Included if validating	Forbidden
Reference in Attribute Value	Not recognized	Included in literal	Forbidden	Forbidden
Occurs as Attribute Value	Not recognized	Forbidden	Forbidden	Notify
Reference in EntityValue	Included in literal	Bypassed	Bypassed	Forbidden
Reference in DTD	Included as PE	Forbidden	Forbidden	Forbidden

4.4.1 Not Recognized

Outside the DTD, the % character has no special significance; thus, what would be parameter entity references in the DTD are not recognized as markup in content. Similarly, the names of unparsed entities are not recognized except when they appear in the value of an appropriately declared attribute.

4.4.2 Included

An entity is **included** when its replacement text is retrieved and processed, in place of the reference itself, as though it were part of the document at the location the reference was recognized. The replacement text may contain both character data and (except for parameter entities) markup, which must be recognized in the usual way, except that the replacement text of entities used to escape markup delimiters (the entities amp, lt, gt, apos, quot) is always treated as data. (The string "AT&T;" expands to "AT&T;" and the remaining ampersand is not recognized as an entity-reference delimiter.) A character reference is **included** when the indicated character is processed in place of the reference itself.

4.4.3 Included If Validating

When an XML processor recognizes a reference to a parsed entity, in order to validate the document, the processor must include its replacement text. If the entity is external, and the processor is not attempting to validate the XML document, the processor may, but need not, include the entity's replacement text. If a non-validating parser does not include the replacement text, it must inform the application that it recognized, but did not read, the entity.

This rule is based on the recognition that the automatic inclusion provided by the SGML and XML entity mechanism, primarily designed to support modularity in authoring, is not necessarily appropriate for other applications, in particular document browsing. Browsers, for example, when encountering an external parsed entity reference, might choose to provide a visual indication of the entity's presence and retrieve it for display only on demand.

4.4.4 Forbidden

The following are forbidden, and constitute fatal errors:

- the appearance of a reference to an unparsed entity.
- the appearance of any character or general-entity reference in the DTD except within an EntityValue or AttValue.
- a reference to an external entity in an attribute value.

4.4.5 Included in Literal

When an entity reference appears in an attribute value, or a parameter entity reference appears in a literal entity value, its replacement text is processed in place of the reference itself as though it were part of the document at the location the reference was recognized, except that a single or double quote character in the replacement text is always treated as a normal data character and will not terminate the literal. For example, this is well-formed:

```
<!ENTITY % YN '"Yes"' >
<!ENTITY WhatHeSaid "He said &YN;" >
```

while this is not:

```
<!ENTITY EndAttr "27'" >
<element attribute='a-&EndAttr;>
```

4.4.6 Notify

When the name of an unparsed entity appears as a token in the value of an attribute of declared type ENTITY or ENTITIES, a validating processor must inform the application of the system and public (if any) identifiers for both the entity and its associated notation.

4.4.7 Bypassed

When a general entity reference appears in the EntityValue in an entity declaration, it is bypassed and left as is.

4.4.8 Included as PE

Just as with external parsed entities, parameter entities need only be included if validating. When a parameter-entity reference is recognized in the DTD and included, its replacement text is enlarged by the attachment of one leading and one following space (#x20) character; the intent is to constrain the replacement text of parameter entities to contain an integral number of grammatical tokens in the DTD.

4.5 Construction of Internal Entity Replacement Text

In discussing the treatment of internal entities, it is useful to distinguish two forms of the entity's value. The **literal entity value** is the quoted string actually present in the entity declaration, corresponding to the non-terminal EntityValue. The **replacement text** is the content of the entity, after replacement of character references and parameter-entity references.

The literal entity value as given in an internal entity declaration (EntityValue) may contain character, parameter-entity, and general-entity references. Such references must be contained entirely within the literal entity value. The actual replacement text that is included as described above must contain the *replacement text* of any parameter entities referred to, and must contain the character referred to, in place of any character references in the literal entity value; however, general-entity references must be left as-is, unexpanded. For example, given the following declarations:

```
<!ENTITY % pub    "&#xc9;ditions Gallimard" >
<!ENTITY   rights "All rights reserved" >
<!ENTITY   book   "La Peste: Albert Camus,
&#xA9; 1947 %pub;. &rights;" >
```

then the replacement text for the entity "book" is:

```
La Peste: Albert Camus,
© 1947 Éditions Gallimard. &rights;
```

The general-entity reference "&rights;" would be expanded should the reference "&book;" appear in the document's content or an attribute value.

These simple rules may have complex interactions; for a detailed discussion of a difficult example, see "D. Expansion of Entity and Character References".

4.6 Predefined Entities

Entity and character references can both be used to **escape** the left angle bracket, ampersand, and other delimiters. A set of general entities (amp, lt, gt, apos, quot) is specified for this purpose. Numeric character references may also be used; they are expanded immediately when recognized and must be treated as character data, so the numeric character references "<" and "&" may be used to escape < and & when they occur in character data.

All XML processors must recognize these entities whether they are declared or not. For interoperability, valid XML documents should declare these entities, like any others, before using them. If the entities in question are declared, they must be declared as internal entities whose replacement text is the single character being escaped or a character reference to that character, as shown below.

```
<!ENTITY lt     "&#38;#60;">
<!ENTITY gt     "&#62;">
<!ENTITY amp    "&#38;#38;">
<!ENTITY apos   "&#39;">
<!ENTITY quot   "&#34;">
```

Note that the < and & characters in the declarations of "lt" and "amp" are doubly escaped to meet the requirement that entity replacement be well-formed.

4.7 Notation Declarations

Notations identify by name the format of unparsed entities, the format of elements which bear a notation attribute, or the application to which a processing instruction is addressed. **Notation declarations** provide a name for the notation, for use in entity and attribute-list declarations and in attribute specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.

Notation Declarations

```
[82] NotationDecl ::= '<!NOTATION' S Name S (ExternalID | PublicID) S? '>'
[83] PublicID     ::= 'PUBLIC' S PubidLiteral
```

XML processors must provide applications with the name and external identifier(s) of any notation declared and referred to in an attribute value, attribute definition, or entity declaration. They may additionally resolve the external identifier into the system identifier, file name, or other information needed to allow the application to call a processor for data in the notation described. (It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.)

4.8 Document Entity

The **document entity** serves as the root of the entity tree and a starting-point for an XML processor. This specification does not specify how the document entity is to be located by an XML processor; unlike other entities, the document entity has no name and might well appear on a processor input stream without any identification at all.

5. Conformance

5.1 Validating and Non-Validating Processors

Conforming XML processors fall into two classes: validating and non-validating. Validating and non-validating processors alike must report violations of this specification's well-formedness constraints in the content of the document entity and any other parsed entities that they read.

Validating processors must report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification. To accomplish this, validating XML processors must read and process the entire DTD and all external parsed entities referenced in the document.

Non-validating processors are required to check only the document entity, including the entire internal DTD subset, for well-formedness. While they are not required to check the document for validity, they are required to **process** all the declarations they read in the internal DTD subset and in any parameter entity that they read, up to the first reference to a parameter entity that they do *not* read; that is to say, they must use the information in those declarations to normalize attribute values, include the replacement text of internal entities, and supply default attribute values. They must not process entity declarations or attribute-list declarations encountered after a reference to a parameter entity that is not read, since the entity may have contained overriding declarations.

5.2 Using XML Processors

The behavior of a validating XML processor is highly predictable; it must read every piece of a document and report all well-formedness and validity violations. Less is required of a non-validating processor; it need not read any part of the document other than the document entity. This has two effects that may be important to users of XML processors:

- Certain well-formedness errors, specifically those that require reading external entities, may not be detected by a non-validating processor. Examples include the constraints entitled Entity Declared, Parsed Entity, and No Recursion, as well as some of the cases described as forbidden in "4.4 XML Processor Treatment of Entities and References".
- The information passed from the processor to the application may vary, depending on whether the processor reads parameter and external entities. For example, a non-validating processor may not normalize attribute values, include the replacement text of internal entities, or supply default attribute values, where doing so depends on having read declarations in external or parameter entities.

For maximum reliability in interoperating between different XML processors, applications which use non-validating processors should not rely on any behaviors not required of such processors. Applications which require facilities such as the use of default attributes or internal entities which are declared in external entities should use validating XML processors.

6. Notation

The formal grammar of XML is given in this specification using a simple Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one symbol, in the form

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are defined by a regular expression, or with an initial lower case letter otherwise. Literal strings are quoted.

Within the expression on the right-hand side of a rule, the following expressions are used to match strings of one or more characters:

#xN
: where N is a hexadecimal integer, the expression matches the character in ISO/IEC 10646 whose canonical (UCS-4) code value, when interpreted as an unsigned binary number, has the value indicated. The number of leading zeros in the #xN form is insignificant; the number of leading zeros in the corresponding code value is governed by the character encoding in use and is not significant for XML.

[a-zA-Z], [#xN-#xN]
: matches any character with a value in the range(s) indicated (inclusive).

[^a-z], [^#xN-#xN]
: matches any character with a value *outside* the range indicated.

[^abc], [^#xN#xN#xN]
: matches any character with a value not among the characters given.

"string"
: matches a literal string matching that given inside the double quotes.

'string'
: matches a literal string matching that given inside the single quotes.

These symbols may be combined to match more complex patterns as follows, where A and B represent simple expressions:

(expression)
: expression is treated as a unit and may be combined as described in this list.

A?
: matches A or nothing; optional A.

A B
: matches A followed by B.

A | B
: matches A or B but not both.

A - B
: matches any string that matches A but does not match B.

A+
: matches one or more occurrences of A.

A*
: matches zero or more occurrences of A.

Other notations used in the productions are:

/* ... */
: comment.

[wfc: ...]
: well-formedness constraint; this identifies by name a constraint on well-formed documents associated with a production.

[vc: ...]
: validity constraint; this identifies by name a constraint on valid documents associated with a production.

Appendices

A. References

A.1 Normative References

IANA
(Internet Assigned Numbers Authority) *Official Names for Character Sets*, ed. Keld Simonsen et al. See ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets.

IETF RFC 1766
IETF (Internet Engineering Task Force). *RFC 1766: Tags for the Identification of Languages*, ed. H. Alvestrand. 1995.

ISO 639
(International Organization for Standardization). *ISO 639:1988 (E). Code for the representation of names of languages.* [Geneva]: International Organization for Standardization, 1988.

ISO 3166
(International Organization for Standardization). *ISO 3166-1:1997 (E). Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes* [Geneva]: International Organization for Standardization, 1997.

ISO/IEC 10646
ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).

Unicode
The Unicode Consortium. *The Unicode Standard, Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press, 1996.

A.2 Other References

Aho/Ullman
Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Reading: Addison-Wesley, 1986, rpt. corr. 1988.

Berners-Lee et al.
Berners-Lee, T., R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax and Semantics*. 1997. (Work in progress; see updates to RFC1738.)

Brüggemann-Klein
Brüggemann-Klein, Anne. *Regular Expressions into Finite Automata*. Extended abstract in I. Simon, Hrsg., LATIN 1992, S. 97-98. Springer-Verlag, Berlin 1992. Full Version in Theoretical Computer Science 120: 197-213, 1993.

Brüggemann-Klein and Wood
Brüggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages*. Universität Freiburg, Institut für Informatik, Bericht 38, Oktober 1991.

Clark
James Clark. Comparison of SGML and XML. See http://www.w3.org/TR/NOTE-sgml-xml-971215.

IETF RFC1738
IETF (Internet Engineering Task Force). *RFC 1738: Uniform Resource Locators (URL)*, ed. T. Berners-Lee, L. Masinter, M. McCahill. 1994.

IETF RFC1808
IETF (Internet Engineering Task Force). *RFC 1808: Relative Uniform Resource Locators*, ed. R. Fielding. 1995.

IETF RFC2141
IETF (Internet Engineering Task Force). *RFC 2141: URN Syntax*, ed. R. Moats. 1997.

ISO 8879
ISO (International Organization for Standardization). *ISO 8879:1986(E). Information processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML).*

First edition -- 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO/IEC 10744
: ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology -- Hypermedia/Time-based Structuring Language (HyTime).* [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe.* [Geneva]: International Organization for Standardization, 1996.

B. Character Classes

Following the characteristics defined in the Unicode standard, characters are classed as base characters (among others, these contain the alphabetic characters of the Latin alphabet, without diacritics), ideographic characters, and combining characters (among others, this class contains most diacritics); these classes combine to form the class of letters. Digits and extenders are also distinguished.

Characters

```
[84]   Letter ::= BaseChar | Ideographic
[85]   BaseChar ::= [#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6]
         | [#x00D8-#x00F6] | [#x00F8-#x00FF] | [#x0100-#x0131]
         | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E]
         | [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5]
         | [#x01FA-#x0217] | [#x0250-#x02A8] | [#x02BB-#x02C1]
         | #x0386 | [#x0388-#x038A] | #x038C | [#x038E-#x03A1]
         | [#x03A3-#x03CE] | [#x03D0-#x03D6] | #x03DA | #x03DC
         | #x03DE | #x03E0 | [#x03E2-#x03F3] | [#x0401-#x040C]
         | [#x040E-#x044F] | [#x0451-#x045C] | [#x045E-#x0481]
         | [#x0490-#x04C4] | [#x04C7-#x04C8] | [#x04CB-#x04CC]
         | [#x04D0-#x04EB] | [#x04EE-#x04F5] | [#x04F8-#x04F9]
         | [#x0531-#x0556] | #x0559 | [#x0561-#x0586]
         | [#x05D0-#x05EA] | [#x05F0-#x05F2] | [#x0621-#x063A]
         | [#x0641-#x064A] | [#x0671-#x06B7] | [#x06BA-#x06BE]
         | [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5
         | [#x06E5-#x06E6] | [#x0905-#x0939] | #x093D
         | [#x0958-#x0961] | [#x0985-#x098C] | [#x098F-#x0990]
         | [#x0993-#x09A8] | [#x09AA-#x09B0] | #x09B2
         | [#x09B6-#x09B9] | [#x09DC-#x09DD] | [#x09DF-#x09E1]
         | [#x09F0-#x09F1] | [#x0A05-#x0A0A] | [#x0A0F-#x0A10]
         | [#x0A13-#x0A28] | [#x0A2A-#x0A30] | [#x0A32-#x0A33]
         | [#x0A35-#x0A36] | [#x0A38-#x0A39] | [#x0A59-#x0A5C]
         | #x0A5E | [#x0A72-#x0A74] | [#x0A85-#x0A8B] | #x0A8D
         | [#x0A8F-#x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0]
         | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0
         | [#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28]
         | [#x0B2A-#x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39]
         | #x0B3D | [#x0B5C-#x0B5D] | [#x0B5F-#x0B61]
         | [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-#x0B95]
         | [#x0B99-#x0B9A] | #x0B9C | [#x0B9E-#x0B9F]
         | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA] | [#x0BAE-#x0BB5]
         | [#x0BB7-#x0BB9] | [#x0C05-#x0C0C] | [#x0C0E-#x0C10]
         | [#x0C12-#x0C28] | [#x0C2A-#x0C33] | [#x0C35-#x0C39]
         | [#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90]
         | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3] | [#x0CB5-#x0CB9]
         | #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C]
         | [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39]
         | [#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30
         | [#x0E32-#x0E33] | [#x0E40-#x0E45] | [#x0E81-#x0E82]
         | #x0E84 | [#x0E87-#x0E88] | #x0E8A | #x0E8D
         | [#x0E94-#x0E97] | [#x0E99-#x0E9F] | [#x0EA1-#x0EA3]
         | #x0EA5 | #x0EA7 | [#x0EAA-#x0EAB] | [#x0EAD-#x0EAE]
         | #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD | [#x0EC0-#x0EC4]
         | [#x0F40-#x0F47] | [#x0F49-#x0F69] | [#x10A0-#x10C5]
         | [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103]
         | [#x1105-#x1107] | #x1109 | [#x110B-#x110C]
         | [#x110E-#x1112] | #x113C | #x113E | #x1140 | #x114C
         | #x114E | #x1150 | [#x1154-#x1155] | #x1159
         | [#x115F-#x1161] | #x1163 | #x1165 | #x1167 | #x1169
```

```
                          | [#x116D-#x116E] | [#x1172-#x1173] | #x1175 | #x119E
                          | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8]
                          | #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9
                          | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15]
                          | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D]
                          | [#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D
                          | [#x1F5F-#x1F7D] | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC]
                          | #x1FBE | [#x1FC2-#x1FC4] | [#x1FC6-#x1FCC]
                          | [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC]
                          | [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126
                          | [#x212A-#x212B] | #x212E | [#x2180-#x2182]
                          | [#x3041-#x3094] | [#x30A1-#x30FA] | [#x3105-#x312C]
                          | [#xAC00-#xD7A3]
[86] Ideographic ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]
[87] CombiningChar ::= [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486]
                          | [#x0591-#x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD]
                          | #x05BF | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652]
                          | #x0670 | [#x06D6-#x06DC] | [#x06DD-#x06DF]
                          | [#x06E0-#x06E4] | [#x06E7-#x06E8] | [#x06EA-#x06ED]
                          | [#x0901-#x0903] | #x093C | [#x093E-#x094C] | #x094D
                          | [#x0951-#x0954] | [#x0962-#x0963] | [#x0981-#x0983]
                          | #x09BC | #x09BE | #x09BF | [#x09C0-#x09C4]
                          | [#x09C7-#x09C8] | [#x09CB-#x09CD] | #x09D7
                          | [#x09E2-#x09E3] | #x0A02 | #x0A3C | #x0A3E | #x0A3F
                          | [#x0A40-#x0A42] | [#x0A47-#x0A48] | [#x0A4B-#x0A4D]
                          | [#x0A70-#x0A71] | [#x0A81-#x0A83] | #x0ABC
                          | [#x0ABE-#x0AC5] | [#x0AC7-#x0AC9] | [#x0ACB-#x0ACD]
                          | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-#x0B43]
                          | [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-#x0B57]
                          | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-#x0BC8]
                          | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03]
                          | [#x0C3E-#x0C44] | [#x0C46-#x0C48] | [#x0C4A-#x0C4D]
                          | [#x0C55-#x0C56] | [#x0C82-#x0C83] | [#x0CBE-#x0CC4]
                          | [#x0CC6-#x0CC8] | [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6]
                          | [#x0D02-#x0D03] | [#x0D3E-#x0D43] | [#x0D46-#x0D48]
                          | [#x0D4A-#x0D4D] | #x0D57 | #x0E31 | [#x0E34-#x0E3A]
                          | [#x0E47-#x0E4E] | #x0EB1 | [#x0EB4-#x0EB9]
                          | [#x0EBB-#x0EBC] | [#x0EC8-#x0ECD] | [#x0F18-#x0F19]
                          | #x0F35 | #x0F37 | #x0F39 | #x0F3E | #x0F3F
                          | [#x0F71-#x0F84] | [#x0F86-#x0F8B] | [#x0F90-#x0F95]
                          | #x0F97 | [#x0F99-#x0FAD] | [#x0FB1-#x0FB7] | #x0FB9
                          | [#x20D0-#x20DC] | #x20E1 | [#x302A-#x302F] | #x3099
                          | #x309A
[88] Digit ::= [#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9]
                          | [#x0966-#x096F] | [#x09E6-#x09EF] | [#x0A66-#x0A6F]
                          | [#x0AE6-#x0AEF] | [#x0B66-#x0B6F] | [#x0BE7-#x0BEF]
                          | [#x0C66-#x0C6F] | [#x0CE6-#x0CEF] | [#x0D66-#x0D6F]
                          | [#x0E50-#x0E59] | [#x0ED0-#x0ED9] | [#x0F20-#x0F29]
[89] Extender ::= #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46
                          | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E]
                          | [#x30FC-#x30FE]
```

The character classes defined here can be derived from the Unicode character database as follows:

- Name start characters must have one of the categories Ll, Lu, Lo, Lt, Nl.
- Name characters other than Name-start characters must have one of the categories Mc, Me, Mn, Lm, or Nd.
- Characters in the compatibility area (i.e. with character code greater than #xF900 and less than #xFFFE) are not allowed in XML names.
- Characters which have a font or compatibility decomposition (i.e. those with a "compatibility formatting tag" in field 5 of the database -- marked by field 5 beginning with a "<") are not allowed.
- The following characters are treated as name-start characters rather than name characters because the property file classifies them as Alphabetic: [#x02BB-#x02C1], #x0559, #x06E5, #x06E6.
- Characters #x20DD-#x20E0 are excluded (in accordance with Unicode, section 5.14).
- Character #x00B7 is classified as an extender, because the property list so identifies it.
- Character #x0387 is added as a name character, because #x00B7 is its canonical equivalent.

- Characters ':' and '_' are allowed as name-start characters.
- Characters '-' and '.' are allowed as name characters.

C. XML and SGML (Non-Normative)

XML is designed to be a subset of SGML, in that every valid XML document should also be a conformant SGML document. For a detailed comparison of the additional restrictions that XML places on documents beyond those of SGML, see [Clark].

D. Expansion of Entity and Character References (Non-Normative)

This appendix contains some examples illustrating the sequence of entity- and character-reference recognition and expansion, as specified in " 4.4 XML Processor Treatment of Entities and References".

If the DTD contains the declaration

```
<!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>" >
```

then the XML processor will recognize the character references when it parses the entity declaration, and resolve them before storing the following string as the value of the entity "example":

```
<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>
```

A reference in the document to "&example;" will cause the text to be reparsed, at which time the start- and end-tags of the "p" element will be recognized and the three references will be recognized and expanded, resulting in a "p" element with the following content (all data, no delimiters or markup):

```
An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;).
```

A more complex example will illustrate the rules and their effects fully. In the following example, the line numbers are solely for reference.

```
1 <?xml version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60;!ENTITY tricky "error-prone" >' >
6 %xx;
7 ]>
8 <test>This sample shows a &tricky; method.</test>
```

This produces the following:

- in line 4, the reference to character 37 is expanded immediately, and the parameter entity "xx" is stored in the symbol table with the value "%zz;". Since the replacement text is not rescanned, the reference to parameter entity "zz" is not recognized. (And it would be an error if it were, since "zz" is not yet declared.)
- in line 5, the character reference "<" is expanded immediately and the parameter entity "zz" is stored with the replacement text "<!ENTITY tricky "error-prone" >", which is a well-formed entity declaration.
- in line 6, the reference to "xx" is recognized, and the replacement text of "xx" (namely "%zz;") is parsed. The reference to "zz" is recognized in its turn, and its replacement text ("<!ENTITY tricky "error-prone" >") is parsed. The general entity "tricky" has now been declared, with the replacement text "error-prone".
- in line 8, the reference to the general entity "tricky" is recognized, and it is expanded, so the full content of the "test" element is the self-describing (and ungrammatical) string *This sample*

shows a error-prone method.

E. Deterministic Content Models (Non-Normative)

For compatibility, it is required that content models in element type declarations be deterministic.
SGML requires deterministic content models (it calls them "unambiguous"); XML processors built using SGML systems may flag non-deterministic content models as errors.
For example, the content model ((b, c) | (b, d)) is non-deterministic, because given an initial b the parser cannot know which b in the model is being matched without looking ahead to see which element follows the b. In this case, the two references to b can be collapsed into a single reference, making the model read (b, (c | d)). An initial b now clearly matches only a single name in the content model. The parser doesn't need to look ahead to see what follows; either c or d would be accepted.
More formally: a finite state automaton may be constructed from the content model using the standard algorithms, e.g. algorithm 3.5 in section 3.9 of Aho, Sethi, and Ullman [Aho/Ullman]. In many such algorithms, a follow set is constructed for each position in the regular expression (i.e., each leaf node in the syntax tree for the regular expression); if any position has a follow set in which more than one following position is labeled with the same element type name, then the content model is in error and may be reported as an error.
Algorithms exist which allow many but not all non-deterministic content models to be reduced automatically to equivalent deterministic models; see Brüggemann-Klein 1991 [Brüggemann-Klein].

F. Autodetection of Character Encodings (Non-Normative)

The XML encoding declaration functions as an internal label on each entity, indicating which character encoding is in use. Before an XML processor can read the internal label, however, it apparently has to know what character encoding is in use--which is what the internal label is trying to indicate. In the general case, this is a hopeless situation. It is not entirely hopeless in XML, however, because XML limits the general case in two ways: each implementation is assumed to support only a finite set of character encodings, and the XML encoding declaration is restricted in position and content in order to make it feasible to autodetect the character encoding in use in each entity in normal cases. Also, in many cases other sources of information are available in addition to the XML data stream itself. Two cases may be distinguished, depending on whether the XML entity is presented to the processor without, or with, any accompanying (external) information. We consider the first case first.
Because each XML entity not in UTF-8 or UTF-16 format *must* begin with an XML encoding declaration, in which the first characters must be '<?xml', any conforming processor can detect, after two to four octets of input, which of the following cases apply. In reading this list, it may help to know that in UCS-4, '<' is "#x0000003C" and '?' is "#x0000003F", and the Byte Order Mark required of UTF-16 data streams is "#xFEFF".

- 00 00 00 3C: UCS-4, big-endian machine (1234 order)
- 3C 00 00 00: UCS-4, little-endian machine (4321 order)
- 00 00 3C 00: UCS-4, unusual octet order (2143)
- 00 3C 00 00: UCS-4, unusual octet order (3412)
- FE FF: UTF-16, big-endian
- FF FE: UTF-16, little-endian
- 00 3C 00 3F: UTF-16, big-endian, no Byte Order Mark (and thus, strictly speaking, in error)
- 3C 00 3F 00: UTF-16, little-endian, no Byte Order Mark (and thus, strictly speaking, in error)
- 3C 3F 78 6D: UTF-8, ISO 646, ASCII, some part of ISO 8859, Shift-JIS, EUC, or any other 7-bit, 8-bit, or mixed-width encoding which ensures that the characters of ASCII have their normal positions, width, and values; the actual encoding declaration must be read to detect which of these applies, but since all of these encodings use the same bit patterns for the ASCII characters, the encoding declaration itself may be read reliably
- 4C 6F A7 94: EBCDIC (in some flavor; the full encoding declaration must be read to tell which code page is in use)

- other: UTF-8 without an encoding declaration, or else the data stream is corrupt, fragmentary, or enclosed in a wrapper of some kind

This level of autodetection is enough to read the XML encoding declaration and parse the character-encoding identifier, which is still necessary to distinguish the individual members of each family of encodings (e.g. to tell UTF-8 from 8859, and the parts of 8859 from each other, or to distinguish the specific EBCDIC code page in use, and so on).

Because the contents of the encoding declaration are restricted to ASCII characters, a processor can reliably read the entire encoding declaration as soon as it has detected which family of encodings is in use. Since in practice, all widely used character encodings fall into one of the categories above, the XML encoding declaration allows reasonably reliable in-band labeling of character encodings, even when external sources of information at the operating-system or transport-protocol level are unreliable.

Once the processor has detected the character encoding in use, it can act appropriately, whether by invoking a separate input routine for each case, or by calling the proper conversion function on each character of input.

Like any self-labeling system, the XML encoding declaration will not work if any software changes the entity's character set or encoding without updating the encoding declaration. Implementors of character-encoding routines should be careful to ensure the accuracy of the internal and external information used to label the entity.

The second possible case occurs when the XML entity is accompanied by encoding information, as in some file systems and some network protocols. When multiple sources of information are available, their relative priority and the preferred method of handling conflict should be specified as part of the higher-level protocol used to deliver XML. Rules for the relative priority of the internal label and the MIME-type label in an external header, for example, should be part of the RFC document defining the text/xml and application/xml MIME types. In the interests of interoperability, however, the following rules are recommended.

- If an XML entity is in a file, the Byte-Order Mark and encoding-declaration PI are used (if present) to determine the character encoding. All other heuristics and sources of information are solely for error recovery.
- If an XML entity is delivered with a MIME type of text/xml, then the charset parameter on the MIME type determines the character encoding method; all other heuristics and sources of information are solely for error recovery.
- If an XML entity is delivered with a MIME type of application/xml, then the Byte-Order Mark and encoding-declaration PI are used (if present) to determine the character encoding. All other heuristics and sources of information are solely for error recovery.

These rules apply only in the absence of protocol-level documentation; in particular, when the MIME types text/xml and application/xml are defined, the recommendations of the relevant RFC will supersede these rules.

G. W3C XML Working Group (Non-Normative)

This specification was prepared and approved for publication by the W3C XML Working Group (WG). WG approval of this specification does not necessarily imply that all WG members voted for its approval. The current and former members of the XML WG are:

Jon Bosak, Sun (Chair); James Clark (Technical Lead); Tim Bray, Textuality and Netscape (XML Co-editor); Jean Paoli, Microsoft (XML Co-editor); C. M. Sperberg-McQueen, U. of Ill. (XML Co-editor); Dan Connolly, W3C (W3C Liaison); Paula Angerstein, Texcel; Steve DeRose, INSO; Dave Hollander, HP; Eliot Kimber, ISOGEN; Eve Maler, ArborText; Tom Magliery, NCSA; Murray Maloney, Muzmo and Grif; Makoto Murata, Fuji Xerox Information Systems; Joel Nava, Adobe; Conleth O'Connell, Vignette; Peter Sharpe, SoftQuad; John Tigue, DataChannel

Appendix M

REC-DOM-Level-1-19981001

Document Object Model (DOM) Level 1 Specification

Version 1.0

W3C Recommendation *1 October, 1998*

This version
http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001
http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/DOM.ps
http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/DOM.pdf
http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/DOM.tgz
http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/DOM.zip
http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/DOM.txt

Latest version
http://www.w3.org/TR/REC-DOM-Level-1

Previous versions
http://www.w3.org/TR/1998/PR-DOM-Level-1-19980818
http://www.w3.org/TR/1998/WD-DOM-19980720
http://www.w3.org/TR/1998/WD-DOM-19980416
http://www.w3.org/TR/WD-DOM-19980318
http://www.w3.org/TR/WD-DOM-971209
http://www.w3.org/TR/WD-DOM-971009

WG Chair
Lauren Wood, *SoftQuad, Inc.*

Editors
Vidur Apparao, *Netscape*
Steve Byrne, *Sun*
Mike Champion, *ArborText*
Scott Isaacs, *Microsoft*
Ian Jacobs, *W3C*
Arnaud Le Hors, *W3C*
Gavin Nicol, *Inso EPS*
Jonathan Robie, *Texcel Research*
Robert Sutor, *IBM*
Chris Wilson, *Microsoft*
Lauren Wood, *SoftQuad, Inc.*

Principal Contributors
Vidur Apparao, *Netscape*
Steve Byrne, *Sun (until November 1997)*
Mike Champion, *ArborText, Inc.*

Scott Isaacs, *Microsoft (until January, 1998)*
Arnaud Le Hors, *W3C*
Gavin Nicol, *Inso EPS*
Jonathan Robie, *Texcel Research*
Peter Sharpe, *SoftQuad, Inc.*
Bill Smith, *Sun (after November 1997)*
Jared Sorensen, *Novell*
Robert Sutor, *IBM*
Ray Whitmer, *iMall*
Chris Wilson, *Microsoft (after January, 1998)*

Status of this document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

The authors of this document are the DOM Working Group members, different chapters may have different editors.

Comments on this document should be sent to the public mailing list www-dom@w3.org.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

Errata

The list of known errors in this document is found at http://www.w3.org/DOM/updates/REC-DOM-Level-1-19981001-errata.html.

Available Languages

The English version of this specification is the only normative version. However, for translations in other languages see http://www.w3.org/DOM/updates/REC-DOM-Level-1-translations.html.

Abstract

This specification defines the Document Object Model Level 1, a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. Vendors can support the DOM as an interface to their proprietary data structures and APIs, and content authors can write to the standard DOM interfaces rather than product-specific APIs, thus increasing interoperability on the Web.

The goal of the DOM specification is to define a programmatic interface for XML and HTML. The DOM Level 1 specification is separated into two parts: Core and HTML. The Core DOM Level 1 section provides a low-level set of fundamental interfaces that can represent any structured document, as well as defining extended interfaces for representing an XML document. These extended XML interfaces need not be implemented by a DOM implementation that only provides access to HTML documents; all of the fundamental interfaces in the Core section must be implemented. A compliant DOM implementation that implements the extended XML interfaces is required to also implement the fundamental Core interfaces, but not the HTML interfaces. The HTML Level 1 section provides additional, higher-level interfaces that are used with the fundamental interfaces defined in the Core Level 1 section to provide a more convenient view of an HTML document. A compliant implementation of the HTML DOM implements all of the fundamental Core interfaces as well as the HTML interfaces.

Table of contents

Table of contents

Expanded Table of Contents

Copyright Notice

Copyright Notice

What is the Document Object Model?

Editors
Jonathan Robie, Texcel Research

Introduction

The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many different kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions - in particular, the DOM interfaces for the XML internal and external subsets have not yet been specified.

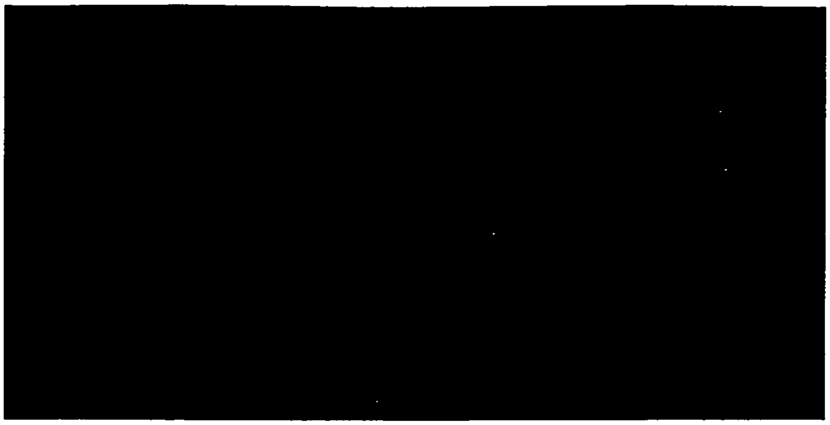
As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The DOM is designed to be used with any programming language. In order to provide a precise, language-independent specification of the DOM interfaces, we have chosen to define the specifications in OMG IDL, as defined in the CORBA 2.2 specification. In addition to the OMG IDL specification, we provide language bindings for Java and ECMAScript (an industry-standard scripting language based on JavaScript and JScript).
Note: OMG IDL is used only as a language-independent and implementation-neutral way to specify interfaces. Various other IDLs could have been used. In general, IDLs are designed for specific computing environments. The Document Object Model can be implemented in any computing environment, and does not require the object binding runtimes generally associated with such IDLs.

What the Document Object Model is

The DOM is a programming API for documents. It closely resembles the structure of the documents it models. For instance, consider this table, taken from an HTML document:

```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>
```

The DOM represents this table like this:



DOM representation of the example table

In the DOM, documents have a logical structure which is very much like a tree; to be more precise, it is like a "forest" or "grove", which can contain more than one tree. However, the DOM does not specify that documents must be *implemented* as a tree or a grove, nor does it specify how the relationships among objects be implemented. The DOM is a logical model that may be implemented in any convenient manner. In this specification, we use the term *structure model* to describe the tree-like representation of a document; we specifically avoid terms like "tree" or "grove" in order to avoid implying a particular implementation. One important property of DOM structure models is *structural isomorphism*: if any two Document Object Model implementations are used to create a representation of the same document, they will create the same structure model, with precisely the same objects and relationships.

The name "Document Object Model" was chosen because it is an "object model" in the traditional object oriented design sense: documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed. In other words, the nodes in the above diagram do not represent a data structure, they represent objects, which have functions and identity. As an object model, the DOM identifies:

- the interfaces and objects used to represent and manipulate a document
- the semantics of these interfaces and objects - including both behavior and attributes
- the relationships and collaborations among these interfaces and objects

The structure of SGML documents has traditionally been represented by an abstract data model, not by an object model. In an abstract data model, the model is centered around the data. In object oriented programming languages, the data itself is encapsulated in objects that hide the data, protecting it from direct external manipulation. The functions associated with these objects determine how the objects may be manipulated, and they are part of the object model.

The Document Object Model currently consists of two parts, DOM Core and DOM HTML. The DOM Core represents the functionality used for XML documents, and also serves as the basis for DOM HTML. A compliant implementation of the DOM must implement all of the fundamental interfaces in the Core chapter with the semantics as defined. Further, it must implement at least one of the HTML DOM and the

extended (XML) interfaces with the semantics as defined.

What the Document Object Model is not

This section is designed to give a more precise understanding of the DOM by distinguishing it from other systems that may seem to be like it.

- Although the Document Object Model was strongly influenced by "Dynamic HTML", in Level 1, it does not implement all of "Dynamic HTML". In particular, events have not yet been defined. Level 1 is designed to lay a firm foundation for this kind of functionality by providing a robust, flexible model of the document itself.
- The Document Object Model is not a binary specification. DOM programs written in the same language will be source code compatible across platforms, but the DOM does not define any form of binary interoperability.
- The Document Object Model is not a way of persisting objects to XML or HTML. Instead of specifying how objects may be represented in XML, the DOM specifies how XML and HTML documents are represented as objects, so that they may be used in object oriented programs.
- The Document Object Model is not a set of data structures, it is an object model that specifies interfaces. Although this document contains diagrams showing parent/child relationships, these are logical relationships defined by the programming interfaces, not representations of any particular internal data structures.

  The Document Object Model does not define "the true inner semantics" of XML or HTML. The semantics of those languages are defined by W3C Recommendations for these languages. The DOM is a programming model designed to respect these semantics. The DOM does not have any ramifications for the way you write XML and HTML documents; any document that can be written in these languages can be represented in the DOM.
- The Document Object Model, despite its name, is not a competitor to the Component Object Model (COM). COM, like CORBA, is a language independent way to specify interfaces and objects; the DOM is a set of interfaces and objects designed for managing HTML and XML documents. The DOM may be implemented using language-independent systems like COM or CORBA; it may also be implemented using language-specific bindings like the Java or ECMAScript bindings specified in this document.

Where the Document Object Model came from

The DOM originated as a specification to allow JavaScript scripts and Java programs to be portable among Web browsers. "Dynamic HTML" was the immediate ancestor of the Document Object Model, and it was originally thought of largely in terms of browsers. However, when the DOM Working Group was formed at W3C, it was also joined by vendors in other domains, including HTML or XML editors and document repositories. Several of these vendors had worked with SGML before XML was developed; as a result, the DOM has been influenced by SGML Groves and the HyTime standard. Some of these vendors had also developed their own object models for documents in order to provide an API for SGML/XML editors or document repositories, and these object models have also influenced the DOM.

Entities and the DOM Core

In the fundamental DOM interfaces, there are no objects representing entities. Numeric character references, and references to the pre-defined entities in HTML and XML, are replaced by the single character that makes up the entity's replacement. For example, in:

```
<p>This is a dog &amp; a cat</p>
```

the "&" will be replaced by the character "&", and the text in the P element will form a single continuous sequence of characters. Since numeric character references and pre-defined entities are not recognized as such in CDATA sections, or the SCRIPT and STYLE elements in HTML, they are not replaced by the single character they appear to refer to. If the example above were enclosed in a CDATA section, the "&" would not be replaced by "&"; neither would the <p> be recognized as a start tag. The representation of general entities, both internal and external, are defined within the extended (XML) interfaces of the Level 1 specification.

Note: When a DOM representation of a document is serialized as XML or HTML text, applications will need to check each character in text data to see if it needs to be escaped using a numeric or pre-defined entity. Failing to do so could result in invalid HTML or XML. Also, implementations should be aware of the fact that serialization into a character encoding ("charset") that does not fully cover ISO 10646 may fail if there are characters in markup or CDATA sections that are not present in the encoding.

DOM Interfaces and DOM Implementations

The DOM specifies interfaces which may be used to manage XML or HTML documents. It is important to realize that these interfaces are an abstraction - much like "abstract base classes" in C++, they are a means of specifying a way to access and manipulate an application's internal representation of a document. Interfaces do not imply a particular concrete implementation. Each DOM application is free to maintain documents in any convenient representation, as long as the interfaces shown in this specification are supported. Some DOM implementations will be existing programs that use the DOM interfaces to access software written long before the DOM specification existed. Therefore, the DOM is designed to avoid implementation dependencies; in particular,

1. Attributes defined in the IDL do not imply concrete objects which must have specific data members - in the language bindings, they are translated to a pair of get()/set() functions, not to a data member. (Read-only functions have only a get() function in the language bindings).
2. DOM applications may provide additional interfaces and objects not found in this specification and still be considered DOM compliant.
3. Because we specify interfaces and not the actual objects that are to be created, the DOM can not know what constructors to call for an implementation. In general, DOM users call the createXXX() methods on the Document class to create document structures, and DOM implementations create their own internal representations of these structures in their implementations of the createXXX() functions.

Limitations of Level 1

The DOM Level 1 specification is intentionally limited to those methods needed to represent and manipulate document structure and content. The plan is for future Levels of the DOM specification to provide:

1. A structure model for the internal subset and the external subset.
2. Validation against a schema.
3. Control for rendering documents via style sheets.
4. Access control.
5. Thread-safety.
6. Events.

1. Document Object Model (Core) Level 1

Editors

Mike Champion, ArborText (from November 20, 1997)
Steve Byrne, JavaSoft (until November 19, 1997)
Gavin Nicol, Inso EPS
Lauren Wood, SoftQuad, Inc.

1.1. Overview of the DOM Core Interfaces

This section defines a minimal set of objects and interfaces for accessing and manipulating document objects. The functionality specified in this section (the *Core* functionality) should be sufficient to allow software developers and web script authors to access and manipulate parsed HTML and XML content inside conforming products. The DOM Core API also allows population of a `Document` [p.22] object using only DOM API calls; creating the skeleton `Document` [p.22] and saving it persistently is left to the product that implements the DOM API.

1.1.1. The DOM Structure Model

The DOM presents documents as a hierarchy of `Node` [p.25] objects that also implement other, more specialized interfaces. Some types of nodes may have child nodes of various types, and others are leaf nodes that cannot have anything below them in the document structure. The node types, and which node types they may have as children, are as follows:

- `Document` [p.22] -- `Element` [p.38] (maximum of one), `ProcessingInstruction` [p.46] , `Comment` [p.43] , `DocumentType` [p.44]
- `DocumentFragment` [p.21] -- `Element` [p.38] , `ProcessingInstruction` [p.46] , `Comment` [p.43] , `Text` [p.42] , `CDATASection` [p.43] , `EntityReference` [p.46]
- `DocumentType` [p.44] -- no children
- `EntityReference` [p.46] -- `Element` [p.38] , `ProcessingInstruction` [p.46] , `Comment` [p.43] , `Text` [p.42] , `CDATASection` [p.43] , `EntityReference` [p.46]
- `Element` [p.38] -- `Element` [p.38] , `Text` [p.42] , `Comment` [p.43] , `ProcessingInstruction` [p.46] , `CDATASection` [p.43] , `EntityReference` [p.46]
- `Attr` [p.37] -- `Text` [p.42] , `EntityReference` [p.46]
- `ProcessingInstruction` [p.46] -- no children
- `Comment` [p.43] -- no children
- `Text` [p.42] -- no children
- `CDATASection` [p.43] -- no children
- `Entity` [p.45] -- `Element` [p.38] , `ProcessingInstruction` [p.46] , `Comment` [p.43] , `Text` [p.42] , `CDATASection` [p.43] , `EntityReference` [p.46]
- `Notation` [p.44] -- no children

The DOM also specifies a `NodeList` [p.32] interface to handle ordered lists of `Node` [p.25] s, such as the children of a `Node` [p.25] , or the elements returned by the `Element.getElementsByTagName` method, and also a `NamedNodeMap` [p.32] interface to handle unordered sets of nodes referenced by their name attribute, such as the attributes of an `Element` [p.38] . `NodeList` [p.32] s and `NamedNodeMap` [p.32] s in the DOM are "live", that is, changes to the underlying document structure are reflected in all relevant `NodeList` [p.32] s and `NamedNodeMap` [p.32] s. For example, if a DOM user gets a `NodeList` [p.32] object containing the children of an `Element` [p.38] , then subsequently adds more children to that element (or removes children, or modifies them), those changes are automatically reflected in the `NodeList` [p.32] without further action on the user's part. Likewise changes to a `Node` [p.25] in the tree are reflected in all references to that `Node` [p.25] in `NodeList` [p.32] s and `NamedNodeMap` [p.32] s.

1.1.2. Memory Management

Most of the APIs defined by this specification are *interfaces* rather than classes. That means that an actual implementation need only expose methods with the defined names and specified operation, not actually implement classes that correspond directly to the interfaces. This allows the DOM APIs to be implemented as a thin veneer on top of legacy applications with their own data structures, or on top of newer applications with different class hierarchies. This also means that ordinary constructors (in the Java or C++ sense) cannot be used to create DOM objects, since the underlying objects to be constructed may have little relationship to the DOM interfaces. The conventional solution to this in object-oriented design is to define *factory* methods that create instances of objects that implement the various interfaces. In the DOM Level 1, objects implementing some interface "X" are created by a "createX()" method on the `Document` [p.22] interface; this is because all DOM objects live in the context of a specific Document.

The DOM Level 1 API does *not* define a standard way to create `DOMImplementation` [p.20] or `Document` [p.22] objects; actual DOM implementations must provide some proprietary way of bootstrapping these DOM interfaces, and then all other objects can be built from the Create methods on `Document` [p.22] (or by various other convenience methods).

The Core DOM APIs are designed to be compatible with a wide range of languages, including both general-user scripting languages and the more challenging languages used mostly by professional programmers. Thus, the DOM APIs need to operate across a variety of memory management philosophies, from language platforms that do not expose memory management to the user at all, through those (notably Java) that provide explicit constructors but provide an automatic garbage collection mechanism to automatically reclaim unused memory, to those (especially C/C++) that generally require the programmer to explicitly allocate object memory, track where it is used, and explicitly free it for re-use. To ensure a consistent API across these platforms, the DOM does not address memory management issues at all, but instead leaves these for the implementation. Neither of the explicit language bindings devised by the DOM Working Group (for ECMAScript and Java) require any memory management methods, but DOM bindings for other languages (especially C or C++) probably will require such support. These extensions will be the responsibility of those adapting the DOM API to a specific language, not the DOM WG.

1.1.3. Naming Conventions

While it would be nice to have attribute and method names that are short, informative, internally consistent, and familiar to users of similar APIs, the names also should not clash with the names in legacy APIs supported by DOM implementations. Furthermore, both OMG IDL and `ECMAScript` have significant limitations in their ability to disambiguate names from different namespaces that makes it difficult to avoid naming conflicts with short, familiar names. So, DOM names tend to be long and quite descriptive in order to be unique across all environments.

The Working Group has also attempted to be internally consistent in its use of various terms, even though these may not be common distinctions in other APIs. For example, we use the method name "remove" when the method changes the structural model, and the method name "delete" when the method gets rid of something inside the structure model. The thing that is deleted is not returned. The thing that is removed may be returned, when it makes sense to return it.

1.1.4. Inheritance vs Flattened Views of the API

The DOM Core APIs present two somewhat different sets of interfaces to an XML/HTML document; one presenting an "object oriented" approach with a hierarchy of inheritance, and a "simplified" view that allows all manipulation to be done via the `Node` [p.25] interface without requiring casts (in Java and other C-like languages) or query interface calls in COM environments. These operations are fairly expensive in Java and COM, and the DOM may be used in performance-critical environments, so we allow significant functionality using just the `Node` [p.25] interface. Because many other users will find the inheritance hierarchy easier to understand than the "everything is a `Node` [p.25] " approach to the DOM, we also support the full higher-level interfaces for those who prefer a more object-oriented API.

In practice, this means that there is a certain amount of redundancy in the API. The Working Group considers the "inheritance" approach the primary view of the API, and the full set of functionality on `Node` [p.25] to be "extra" functionality that users may employ, but that does not eliminate the need for methods on other interfaces that an object-oriented analysis would dictate. (Of course, when the O-O analysis yields an attribute or method that is identical to one on the `Node` [p.25] interface, we don't specify a completely redundant one). Thus, even though there is a generic `nodeName` attribute on the `Node` [p.25] interface, there is still a `tagName` attribute on the `Element` [p.38] interface; these two attributes must contain the same value, but the Working Group considers it worthwhile to support both, given the different constituencies the DOM API must satisfy.

1.1.5. The `DOMString` type

To ensure interoperability, the DOM specifies the `DOMString` type as follows:

- A `DOMString` is a sequence of 16-bit quantities. This may be expressed in IDL terms as:

```
typedef sequence<unsigned short> DOMString;
```

- Applications must encode `DOMString` using UTF-16 (defined in Appendix C.3 of [UNICODE] and Amendment 1 of [ISO-10646]).The UTF-16 encoding was chosen because of its widespread industry practice. Please note that for both HTML and XML, the document character set (and therefore the notation of numeric character references) is based on UCS-4. A single numeric character reference in a source document may therefore in some cases correspond to two array positions in a `DOMString` (a high surrogate and a low surrogate). ***Note:*** *Even though the DOM defines the name of the string type to be `DOMString`, bindings may used different names. For, example for Java, `DOMString` is bound to the `String` type because it also uses UTF-16 as its encoding.*

Note: *As of August 1998, the OMG IDL specification included a `wstring` type. However, that definition did not meet the interoperability criteria of the DOM API since it relied on encoding negotiation to decide the width of a character.*

1.1.6. Case sensitivity in the DOM

The DOM has many interfaces that imply string matching. HTML processors generally assume an uppercase (less often, lowercase) normalization of names for such things as elements, while XML is explicitly case sensitive. For the purposes of the DOM, string matching takes place on a character code by character code basis, on the 16 bit value of a `DOMString`. As such, the DOM assumes that any normalizations will take place in the processor, *before* the DOM structures are built.

This then raises the issue of exactly what normalizations occur. The W3C I18N working group is in the process of defining exactly which normalizations are necessary for applications implementing the DOM.

1.2. Fundamental Interfaces

The interfaces within this section are considered *fundamental*, and must be fully implemented by all conforming implementations of the DOM, including all HTML DOM implementations.

Exception *DOMException*

DOM operations only raise exceptions in "exceptional" circumstances, i.e., when an operation is impossible to perform (either for logical reasons, because data is lost, or because the implementation has become unstable). In general, DOM methods return specific error values in ordinary processing situation, such as out-of-bound errors when using `NodeList` [p.32] .

Implementations may raise other exceptions under other circumstances. For example, implementations may raise an implementation-dependent exception if a `null` argument is passed.

Some languages and object systems do not support the concept of exceptions. For such systems, error conditions may be indicated using native error reporting mechanisms. For some bindings, for example, methods may return error codes similar to those listed in the corresponding method descriptions.

IDL Definition

```
exception DOMException {
  unsigned short   code;
};

// ExceptionCode
const unsigned short      INDEX_SIZE_ERR     = 1;
const unsigned short      DOMSTRING_SIZE_ERR = 2;
const unsigned short      HIERARCHY_REQUEST_ERR = 3;
const unsigned short      WRONG_DOCUMENT_ERR = 4;
const unsigned short      INVALID_CHARACTER_ERR = 5;
const unsigned short      NO_DATA_ALLOWED_ERR = 6;
const unsigned short      NO_MODIFICATION_ALLOWED_ERR = 7;
const unsigned short      NOT_FOUND_ERR      = 8;
const unsigned short      NOT_SUPPORTED_ERR  = 9;
const unsigned short      INUSE_ATTRIBUTE_ERR = 10;
```

Definition group *ExceptionCode*

An integer indicating the type of error generated.
Defined Constants

INDEX_SIZE_ERR	If index or size is negative, or greater than the allowed value
DOMSTRING_SIZE_ERR	If the specified range of text does not fit into a DOMString
HIERARCHY_REQUEST_ERR	If any node is inserted somewhere it doesn't belong
WRONG_DOCUMENT_ERR	If a node is used in a different document than the one that created it (that doesn't support it)
INVALID_CHARACTER_ERR	If an invalid character is specified, such as in a name.
NO_DATA_ALLOWED_ERR	If data is specified for a node which does not support data
NO_MODIFICATION_ALLOWED_ERR	If an attempt is made to modify an object where modifications are not allowed
NOT_FOUND_ERR	If an attempt was made to reference a node in a context where it does not exist
NOT_SUPPORTED_ERR	If the implementation does not support the type of object requested
INUSE_ATTRIBUTE_ERR	If an attempt is made to add an attribute that is already inuse elsewhere

Interface *DOMImplementation*

The `DOMImplementation` interface provides a number of methods for performing operations that are independent of any particular instance of the document object model.

The DOM Level 1 does not specify a way of creating a document instance, and hence document creation is an operation specific to an implementation. Future Levels of the DOM specification are expected to provide methods for creating documents directly.
IDL Definition

```
interface DOMImplementation {
  boolean                   hasFeature(in DOMString feature,
                                       in DOMString version);
};
```

Methods

`hasFeature`

Test if the DOM implementation implements a specific feature.

Parameters

`feature`	The package name of the feature to test. In Level 1, the legal values are "HTML" and "XML" (case-insensitive).
`version`	This is the version number of the package name to test. In Level 1, this is the string "1.0". If the version is not specified, supporting any version of the feature will cause the method to return `true`.

Return Value

`true` if the feature is implemented in the specified version, `false` otherwise.

This method raises no exceptions.

Interface *DocumentFragment*

`DocumentFragment` is a "lightweight" or "minimal" `Document` [p.22] object. It is very common to want to be able to extract a portion of a document's tree or to create a new fragment of a document. Imagine implementing a user command like cut or rearranging a document by moving fragments around. It is desirable to have an object which can hold such fragments and it is quite natural to use a Node for this purpose. While it is true that a `Document` [p.22] object could fulfil this role, a `Document` [p.22] object can potentially be a heavyweight object, depending on the underlying implementation. What is really needed for this is a very lightweight object. `DocumentFragment` is such an object.

Furthermore, various operations -- such as inserting nodes as children of another `Node` [p.25] -- may take `DocumentFragment` objects as arguments; this results in all the child nodes of the `DocumentFragment` being moved to the child list of this node.

The children of a `DocumentFragment` node are zero or more nodes representing the tops of any sub-trees defining the structure of the document. `DocumentFragment` nodes do not need to be well-formed XML documents (although they do need to follow the rules imposed upon well-formed XML parsed entities, which can have multiple top nodes). For example, a `DocumentFragment` might have only one child and that child node could be a `Text` [p.42] node. Such a structure model represents neither an HTML document nor a well-formed XML document.

When a `DocumentFragment` is inserted into a `Document` [p.22] (or indeed any other `Node` [p.25] that may take children) the children of the `DocumentFragment` and not the `DocumentFragment` itself are inserted into the `Node` [p.25] . This makes the `DocumentFragment` very useful when the user wishes to create nodes that are siblings; the

DocumentFragment acts as the parent of these nodes so that the user can use the standard methods from the Node [p.25] interface, such as insertBefore() and appendChild().

IDL Definition

```
interface DocumentFragment : Node {
};
```

Interface *Document*

The Document interface represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data.

Since elements, text nodes, comments, processing instructions, etc. cannot exist outside the context of a Document, the Document interface also contains the factory methods needed to create these objects. The Node [p.25] objects created have a ownerDocument attribute which associates them with the Document within whose context they were created.

IDL Definition

```
interface Document : Node {
  readonly attribute  DocumentType         doctype;
  readonly attribute  DOMImplementation    implementation;
  readonly attribute  Element              documentElement;
  Element                    createElement(in DOMString tagName)
                                  raises(DOMException);
  DocumentFragment           createDocumentFragment();
  Text                       createTextNode(in DOMString data);
  Comment                    createComment(in DOMString data);
  CDATASection               createCDATASection(in DOMString data)
                                       raises(DOMException);
  ProcessingInstruction      createProcessingInstruction(in DOMString target,
                                                  in DOMString data)
                                                  raises(DOMException);
  Attr                       createAttribute(in DOMString name)
                                   raises(DOMException);
  EntityReference            createEntityReference(in DOMString name)
                                           raises(DOMException);
  NodeList                   getElementsByTagName(in DOMString tagname);
};
```

Attributes

doctype

The Document Type Declaration (see DocumentType [p.44]) associated with this document. For HTML documents as well as XML documents without a document type declaration this returns null. The DOM Level 1 does not support editing the Document Type Declaration, therefore docType cannot be altered in any way.

implementation

The DOMImplementation [p.20] object that handles this document. A DOM application may use objects from multiple implementations.

documentElement

This is a convenience attribute that allows direct access to the child node that is the root element of the document. For HTML documents, this is the element with the tagName "HTML".

Methods

`createElement`

Creates an element of the type specified. Note that the instance returned implements the Element interface, so attributes can be specified directly on the returned object.

Parameters

`tagName`	The name of the element type to instantiate. For XML, this is case-sensitive. For HTML, the `tagName` parameter may be provided in any case, but it must be mapped to the canonical uppercase form by the DOM implementation.

Return Value

A new `Element` [p.38] object.

Exceptions

`DOMException` [p.19]

INVALID_CHARACTER_ERR: Raised if the specified name contains an invalid character.

`createDocumentFragment`

Creates an empty `DocumentFragment` [p.21] object.

Return Value

A new `DocumentFragment` [p.21] .

This method has no parameters.

This method raises no exceptions.

`createTextNode`

Creates a `Text` [p.42] node given the specified string.

Parameters

`data`	The data for the node.

Return Value

The new `Text` [p.42] object.

This method raises no exceptions.

`createComment`

Creates a `Comment` [p.43] node given the specified string.

Parameters

`data`	The data for the node.

Return Value

The new `Comment` [p.43] object.

This method raises no exceptions.

`createCDATASection`

Creates a `CDATASection` [p.43] node whose value is the specified string.

Parameters

`data`	The data for the `CDATASection` [p.43] contents.

Return Value

The new `CDATASection` [p.43] object.

Exceptions

`DOMException` [p.19]

NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

`createProcessingInstruction`

Creates a `ProcessingInstruction` [p.46] node given the specified name and data strings.

Parameters

`target`	The target part of the processing instruction.
`data`	The data for the node.

Return Value

The new `ProcessingInstruction` [p.46] object.

Exceptions

`DOMException` [p.19]

INVALID_CHARACTER_ERR: Raised if an invalid character is specified.

NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

`createAttribute`

Creates an `Attr` [p.37] of the given name. Note that the `Attr` [p.37] instance can then be set on an `Element` [p.38] using the `setAttribute` method.

Parameters

`name`	The name of the attribute.

Return Value

A new `Attr` [p.37] object.

Exceptions

`DOMException` [p.19]

INVALID_CHARACTER_ERR: Raised if the specified name contains an invalid character.

`createEntityReference`

Creates an EntityReference object.

Parameters

`name`	The name of the entity to reference.

Return Value

The new `EntityReference` [p.46] object.

Exceptions

`DOMException` [p.19]

INVALID_CHARACTER_ERR: Raised if the specified name contains an invalid character.

NOT_SUPPORTED_ERR: Raised if this document is an HTML document.

`getElementsByTagName`

Returns a `NodeList` [p.32] of all the `Element` [p.38] s with a given tag name in the order in which they would be encountered in a preorder traversal of the `Document` tree.

Parameters

`tagname`	The name of the tag to match on. The special value "*" matches all tags.

Return Value

A new `NodeList` [p.32] object containing all the matched `Element` [p.38] s.

This method raises no exceptions.

Interface *Node*

The `Node` interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree. While all objects implementing the `Node` interface expose methods for dealing with children, not all objects implementing the `Node` interface may have children. For example, `Text` [p.42] nodes may not have children, and adding children to such nodes results in a `DOMException` [p.19] being raised.

The attributes `nodeName`, `nodeValue` and `attributes` are included as a mechanism to get at node information without casting down to the specific derived interface. In cases where there is no obvious mapping of these attributes for a specific `nodeType` (e.g., `nodeValue` for an Element or `attributes` for a Comment), this returns `null`. Note that the specialized interfaces may contain additional and more convenient mechanisms to get and set the relevant information.

IDL Definition

```
interface Node {
  // NodeType
  const unsigned short      ELEMENT_NODE       = 1;
  const unsigned short      ATTRIBUTE_NODE     = 2;
  const unsigned short      TEXT_NODE          = 3;
  const unsigned short      CDATA_SECTION_NODE = 4;
  const unsigned short      ENTITY_REFERENCE_NODE = 5;
  const unsigned short      ENTITY_NODE        = 6;
  const unsigned short      PROCESSING_INSTRUCTION_NODE = 7;
  const unsigned short      COMMENT_NODE       = 8;
  const unsigned short      DOCUMENT_NODE      = 9;
```

```
  const unsigned short      DOCUMENT_TYPE_NODE = 10;
  const unsigned short      DOCUMENT_FRAGMENT_NODE = 11;
  const unsigned short      NOTATION_NODE      = 12;

  readonly attribute  DOMString             nodeName;
           attribute  DOMString             nodeValue;
                                                  // raises(DOMException) on setting
                                                  // raises(DOMException) on retrieval
  readonly attribute  unsigned short        nodeType;
  readonly attribute  Node                  parentNode;
  readonly attribute  NodeList              childNodes;
  readonly attribute  Node                  firstChild;
  readonly attribute  Node                  lastChild;
  readonly attribute  Node                  previousSibling;
  readonly attribute  Node                  nextSibling;
  readonly attribute  NamedNodeMap          attributes;
  readonly attribute  Document              ownerDocument;
  Node                      insertBefore(in Node newChild,
                                         in Node refChild)
                                         raises(DOMException);
  Node                      replaceChild(in Node newChild,
                                         in Node oldChild)
                                         raises(DOMException);
  Node                      removeChild(in Node oldChild)
                                        raises(DOMException);
  Node                      appendChild(in Node newChild)
                                        raises(DOMException);
  boolean                   hasChildNodes();
  Node                      cloneNode(in boolean deep);
};
```

Definition group *NodeType*

An integer indicating which type of node this is.

Defined Constants

ELEMENT_NODE	The node is a `Element` [p.38] .
ATTRIBUTE_NODE	The node is an `Attr` [p.37] .
TEXT_NODE	The node is a `Text` [p.42] node.
CDATA_SECTION_NODE	The node is a `CDATASection` [p.43] .
ENTITY_REFERENCE_NODE	The node is an `EntityReference` [p.46] .
ENTITY_NODE	The node is an `Entity` [p.45] .
PROCESSING_INSTRUCTION_NODE	The node is a `ProcessingInstruction` [p.46] .
COMMENT_NODE	The node is a `Comment` [p.43] .
DOCUMENT_NODE	The node is a `Document` [p.22] .
DOCUMENT_TYPE_NODE	The node is a `DocumentType` [p.44] .
DOCUMENT_FRAGMENT_NODE	The node is a `DocumentFragment` [p.21] .
NOTATION_NODE	The node is a `Notation` [p.44] .

The values of `nodeName`, `nodeValue`, and `attributes` vary according to the node type as follows:

	nodeName	nodeValue	attributes
Element	tagName	null	NamedNodeMap
Attr	name of attribute	value of attribute	null
Text	#text	content of the text node	null
CDATASection	#cdata-section	content of the CDATA Section	null
EntityReference	name of entity referenced	null	null
Entity	entity name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Comment	#comment	content of the comment	null
Document	#document	null	null
DocumentType	document type name	null	null
DocumentFragment	#document-fragment	null	null
Notation	notation name	null	null

Attributes

`nodeName`

The name of this node, depending on its type; see the table above.

`nodeValue`

The value of this node, depending on its type; see the table above.

Exceptions on setting

`DOMException` [p.19]

NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

Exceptions on retrieval

`DOMException` [p.19]

DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

`nodeType`

A code representing the type of the underlying object, as defined above.

`parentNode`

The parent of this node. All nodes, except `Document` [p.22] , `DocumentFragment` [p.21] , and `Attr` [p.37] may have a parent. However, if a node has just been created and not yet added to the tree, or if it has been removed from the tree, this is `null`.

`childNodes`
A `NodeList` [p.32] that contains all children of this node. If there are no children, this is a `NodeList` [p.32] containing no nodes. The content of the returned `NodeList` [p.32] is "live" in the sense that, for instance, changes to the children of the node object that it was created from are immediately reflected in the nodes returned by the `NodeList` [p.32] accessors; it is not a static snapshot of the content of the node. This is true for every `NodeList` [p.32] , including the ones returned by the `getElementsByTagName` method.

`firstChild`
The first child of this node. If there is no such node, this returns `null`.

`lastChild`
The last child of this node. If there is no such node, this returns `null`.

`previousSibling`
The node immediately preceding this node. If there is no such node, this returns `null`.

`nextSibling`
The node immediately following this node. If there is no such node, this returns `null`.

`attributes`
A `NamedNodeMap` [p.32] containing the attributes of this node (if it is an `Element` [p.38]) or `null` otherwise.

`ownerDocument`
The `Document` [p.22] object associated with this node. This is also the `Document` [p.22] object used to create new nodes. When this node is a `Document` [p.22] this is `null`.

Methods

`insertBefore`
Inserts the node `newChild` before the existing child node `refChild`. If `refChild` is `null`, insert `newChild` at the end of the list of children.

If `newChild` is a `DocumentFragment` [p.21] object, all of its children are inserted, in the same order, before `refChild`. If the `newChild` is already in the tree, it is first removed.

Parameters

`newChild`	The node to insert.
`refChild`	The reference node, i.e., the node before which the new node must be inserted.

Return Value
The node being inserted.

Exceptions
`DOMException` [p.19]

HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to insert is one of this node's ancestors.

WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if `refChild` is not a child of this node.

`replaceChild`

Replaces the child node `oldChild` with `newChild` in the list of children, and returns the `oldChild` node. If the `newChild` is already in the tree, it is first removed.

Parameters

`newChild`	The new node to put in the child list.
`oldChild`	The node being replaced in the list.

Return Value

The node replaced.

Exceptions

`DOMException` [p.19]

HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or it the node to put in is one of this node's ancestors.

WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if `oldChild` is not a child of this node.

`removeChild`

Removes the child node indicated by `oldChild` from the list of children, and returns it.

Parameters

`oldChild`	The node being removed.

Return Value

The node removed.

Exceptions

`DOMException` [p.19]

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if `oldChild` is not a child of this node.

`appendChild`

Adds the node `newChild` to the end of the list of children of this node. If the `newChild` is already in the tree, it is first removed.

Parameters

`newChild`	The node to add. If it is a `DocumentFragment` [p.21] object, the entire contents of the document fragment are moved into the child list of this node

Return Value
The node added.
Exceptions
`DOMException` [p.19]

HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of the type of the `newChild` node, or if the node to append is one of this node's ancestors.

WRONG_DOCUMENT_ERR: Raised if `newChild` was created from a different document than the one that created this node.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

`hasChildNodes`

This is a convenience method to allow easy determination of whether a node has any children.

Return Value
`true` if the node has any children, `false` if the node has no children.

This method has no parameters.

This method raises no exceptions.

`cloneNode`

Returns a duplicate of this node, i.e., serves as a generic copy constructor for nodes. The duplicate node has no parent (`parentNode` returns `null`.).

Cloning an `Element` [p.38] copies all attributes and their values, including those generated by the XML processor to represent defaulted attributes, but this method does not copy any text it contains unless it is a deep clone, since the text is contained in a child `Text` [p.42] node. Cloning any other type of node simply returns a copy of this node.

Parameters

`deep`	If `true`, recursively clone the subtree under the specified node; if `false`, clone only the node itself (and its attributes, if it is an `Element` [p.38]).

Return Value
The duplicate node.

This method raises no exceptions.

Interface *NodeList*

The `NodeList` interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented.

The items in the `NodeList` are accessible via an integral index, starting from 0.

IDL Definition

```
interface NodeList {
  Node                      item(in unsigned long index);
  readonly attribute  unsigned long        length;
};
```

Methods

`item`

Returns the `index`th item in the collection. If `index` is greater than or equal to the number of nodes in the list, this returns `null`.

Parameters

`index`	Index into the collection.

Return Value

The node at the `index`th position in the `NodeList`, or `null` if that is not a valid index.

This method raises no exceptions.

Attributes

`length`

The number of nodes in the list. The range of valid child node indices is 0 to `length-1` inclusive.

Interface *NamedNodeMap*

Objects implementing the `NamedNodeMap` interface are used to represent collections of nodes that can be accessed by name. Note that `NamedNodeMap` does not inherit from `NodeList` [p.32] ; `NamedNodeMap`s are not maintained in any particular order. Objects contained in an object implementing `NamedNodeMap` may also be accessed by an ordinal index, but this is simply to allow convenient enumeration of the contents of a `NamedNodeMap`, and does not imply that the DOM specifies an order to these Nodes.

IDL Definition

```
interface NamedNodeMap {
  Node                      getNamedItem(in DOMString name);
  Node                      setNamedItem(in Node arg)
                                         raises(DOMException);
  Node                      removeNamedItem(in DOMString name)
                                            raises(DOMException);
  Node                      item(in unsigned long index);
  readonly attribute  unsigned long        length;
};
```

Methods

`getNamedItem`

Retrieves a node specified by name.

Parameters

`name`	Name of a node to retrieve.

Return Value

A `Node` [p.25] (of any type) with the specified name, or `null` if the specified name did not identify any node in the map.

This method raises no exceptions.

`setNamedItem`

Adds a node using its `nodeName` attribute.

As the `nodeName` attribute is used to derive the name which the node must be stored under, multiple nodes of certain types (those that have a "special" string value) cannot be stored as the names would clash. This is seen as preferable to allowing nodes to be aliased.

Parameters

`arg`	A node to store in a named node map. The node will later be accessible using the value of the `nodeName` attribute of the node. If a node with that name is already present in the map, it is replaced by the new one.

Return Value

If the new `Node` [p.25] replaces an existing node with the same name the previously existing `Node` [p.25] is returned, otherwise `null` is returned.

Exceptions

`DOMException` [p.19]

WRONG_DOCUMENT_ERR: Raised if `arg` was created from a different document than the one that created the `NamedNodeMap`.

NO_MODIFICATION_ALLOWED_ERR: Raised if this `NamedNodeMap` is readonly.

INUSE_ATTRIBUTE_ERR: Raised if `arg` is an `Attr` [p.37] that is already an attribute of another `Element` [p.38] object. The DOM user must explicitly clone `Attr` [p.37] nodes to re-use them in other elements.

`removeNamedItem`

Removes a node specified by name. If the removed node is an `Attr` [p.37] with a default value it is immediately replaced.

Parameters

`name`	The name of a node to remove.

Return Value

The node removed from the map or `null` if no node with such a name exists.

Exceptions

`DOMException` [p.19]

NOT_FOUND_ERR: Raised if there is no node named `name` in the map.

`item`

Returns the `index`th item in the map. If `index` is greater than or equal to the number of nodes in the map, this returns `null`.

Parameters

`index`	Index into the map.

Return Value

The node at the `index`th position in the `NamedNodeMap`, or `null` if that is not a valid index.

This method raises no exceptions.

Attributes

`length`

The number of nodes in the map. The range of valid child node indices is 0 to `length-1` inclusive.

Interface *CharacterData*

The `CharacterData` interface extends Node with a set of attributes and methods for accessing character data in the DOM. For clarity this set is defined here rather than on each object that uses these attributes and methods. No DOM objects correspond directly to `CharacterData`, though `Text` [p.42] and others do inherit the interface from it. All `offsets` in this interface start from 0.

IDL Definition

```
interface CharacterData : Node {
           attribute  DOMString            data;
                                // raises(DOMException) on setting
                                // raises(DOMException) on retrieval
  readonly attribute  unsigned long        length;
  DOMString                 substringData(in unsigned long offset,
                                          in unsigned long count)
                                          raises(DOMException);
  void                      appendData(in DOMString arg)
                                       raises(DOMException);
  void                      insertData(in unsigned long offset,
                                       in DOMString arg)
                                       raises(DOMException);
  void                      deleteData(in unsigned long offset,
                                       in unsigned long count)
                                       raises(DOMException);
  void                      replaceData(in unsigned long offset,
```

```
                                      in unsigned long count,
                                      in DOMString arg)
                                      raises(DOMException);
};
```

Attributes

`data`

The character data of the node that implements this interface. The DOM implementation may not put arbitrary limits on the amount of data that may be stored in a `CharacterData` node. However, implementation limits may mean that the entirety of a node's data may not fit into a single `DOMString`. In such cases, the user may call `substringData` to retrieve the data in appropriately sized pieces.

Exceptions on setting

`DOMException` [p.19]

NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

Exceptions on retrieval

`DOMException` [p.19]

DOMSTRING_SIZE_ERR: Raised when it would return more characters than fit in a `DOMString` variable on the implementation platform.

`length`

The number of characters that are available through `data` and the `substringData` method below. This may have the value zero, i.e., `CharacterData` nodes may be empty.

Methods

`substringData`

Extracts a range of data from the node.

Parameters

`offset`	Start offset of substring to extract.
`count`	The number of characters to extract.

Return Value

The specified substring. If the sum of `offset` and `count` exceeds the `length`, then all characters to the end of the data are returned.

Exceptions

`DOMException` [p.19]

INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of characters in `data`, or if the specified `count` is negative.

DOMSTRING_SIZE_ERR: Raised if the specified range of text does not fit into a `DOMString`.

`appendData`

Append the string to the end of the character data of the node. Upon success, `data` provides access to the concatenation of `data` and the `DOMString` specified.

Parameters

`arg`	The `DOMString` to append.

Exceptions

`DOMException` [p.19]

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

`insertData`

Insert a string at the specified character offset.

Parameters

`offset`	The character offset at which to insert.
`arg`	The `DOMString` to insert.

Exceptions

`DOMException` [p.19]

INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of characters in `data`.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

`deleteData`

Remove a range of characters from the node. Upon success, `data` and `length` reflect the change.

Parameters

`offset`	The offset from which to remove characters.
`count`	The number of characters to delete. If the sum of `offset` and `count` exceeds `length` then all characters from `offset` to the end of the data are deleted.

Exceptions

`DOMException` [p.19]

INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of characters in `data`, or if the specified `count` is negative.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

`replaceData`

Replace the characters starting at the specified character offset with the specified string.

Parameters

`offset`	The offset from which to start replacing.
`count`	The number of characters to replace. If the sum of `offset` and `count` exceeds `length`, then all characters to the end of the data are replaced (i.e., the effect is the same as a `remove` method call with the same range, followed by an `append` method invocation).
`arg`	The `DOMString` with which the range must be replaced.

Exceptions

`DOMException` [p.19]

INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of characters in `data`, or if the specified `count` is negative.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

Interface *Attr*

The `Attr` interface represents an attribute in an `Element` [p.38] object. Typically the allowable values for the attribute are defined in a document type definition.

`Attr` objects inherit the `Node` [p.25] interface, but since they are not actually child nodes of the element they describe, the DOM does not consider them part of the document tree. Thus, the `Node` [p.25] attributes `parentNode`, `previousSibling`, and `nextSibling` have a null value for `Attr` objects. The DOM takes the view that attributes are properties of elements rather than having a separate identity from the elements they are associated with; this should make it more efficient to implement such features as default attributes associated with all elements of a given type. Furthermore, `Attr` nodes may not be immediate children of a `DocumentFragment` [p.21] . However, they can be associated with `Element` [p.38] nodes contained within a `DocumentFragment` [p.21] . In short, users and implementors of the DOM need to be aware that `Attr` nodes have some things in common with other objects inheriting the `Node` [p.25] interface, but they also are quite distinct.

The attribute's effective value is determined as follows: if this attribute has been explicitly assigned any value, that value is the attribute's effective value; otherwise, if there is a declaration for this attribute, and that declaration includes a default value, then that default value is the attribute's effective value; otherwise, the attribute does not exist on this element in the structure model until it has been explicitly added. Note that the `nodeValue` attribute on the `Attr` instance can also be used to retrieve the string version of the attribute's value(s).

In XML, where the value of an attribute can contain entity references, the child nodes of the `Attr` node provide a representation in which entity references are not expanded. These child nodes may be either `Text` [p.42] or `EntityReference` [p.46] nodes. Because the attribute type may be unknown, there are no tokenized attribute values.

IDL Definition

```
interface Attr : Node {
  readonly attribute  DOMString            name;
  readonly attribute  boolean              specified;
           attribute  DOMString            value;
};
```

Attributes

`name`

Returns the name of this attribute.

`specified`

If this attribute was explicitly given a value in the original document, this is `true`; otherwise, it is `false`. Note that the implementation is in charge of this attribute, not the user. If the user changes the value of the attribute (even if it ends up having the same value as the default value) then the `specified` flag is automatically flipped to `true`. To re-specify the attribute as the default value from the DTD, the user must delete the attribute. The implementation will then make a new attribute available with `specified` set to `false` and the default value (if one exists).

In summary:

- If the attribute has an assigned value in the document then `specified` is `true`, and the value is the assigned value.
- If the attribute has no assigned value in the document and has a default value in the DTD, then `specified` is `false`, and the value is the default value in the DTD.
- If the attribute has no assigned value in the document and has a value of #IMPLIED in the DTD, then the attribute does not appear in the structure model of the document.

`value`

On retrieval, the value of the attribute is returned as a string. Character and general entity references are replaced with their values.

On setting, this creates a `Text` [p.42] node with the unparsed contents of the string.

Interface *Element*

By far the vast majority of objects (apart from text) that authors encounter when traversing a document are `Element` nodes. Assume the following XML document:

```
<elementExample id="demo">
  <subelement1/>
  <subelement2><subsubelement/></subelement2>
</elementExample>
```

When represented using DOM, the top node is an `Element` node for "elementExample", which contains two child `Element` nodes, one for "subelement1" and one for "subelement2". "subelement1" contains no child nodes.

Elements may have attributes associated with them; since the `Element` interface inherits from `Node` [p.25], the generic `Node` [p.25] interface method `getAttributes` may be used to retrieve the set of all attributes for an element. There are methods on the `Element` interface to retrieve either an `Attr` [p.37] object by name or an attribute value by name. In XML, where an attribute value may contain entity references, an `Attr` [p.37] object should be retrieved to examine the possibly fairly complex sub-tree representing the attribute value. On the other hand, in HTML, where all attributes have simple string values, methods to directly access an attribute value can safely be used as a convenience.

IDL Definition

```
interface Element : Node {
  readonly attribute  DOMString           tagName;
  DOMString                  getAttribute(in DOMString name);
  void                       setAttribute(in DOMString name,
                                          in DOMString value)
                                          raises(DOMException);
  void                       removeAttribute(in DOMString name)
                                             raises(DOMException);
  Attr                       getAttributeNode(in DOMString name);
  Attr                       setAttributeNode(in Attr newAttr)
                                              raises(DOMException);
  Attr                       removeAttributeNode(in Attr oldAttr)
                                                 raises(DOMException);
  NodeList                   getElementsByTagName(in DOMString name);
  void                       normalize();
};
```

Attributes

`tagName`

The name of the element. For example, in:

```
<elementExample id="demo">
        ...
</elementExample> ,
```

`tagName` has the value `"elementExample"`. Note that this is case-preserving in XML, as are all of the operations of the DOM. The HTML DOM returns the `tagName` of an HTML element in the canonical uppercase form, regardless of the case in the source HTML document.

Methods

`getAttribute`

Retrieves an attribute value by name.

Parameters

`name`	The name of the attribute to retrieve.

Return Value

The `Attr` [p.37] value as a string, or the empty string if that attribute does not have a specified or default value.

This method raises no exceptions.

`setAttribute`

Adds a new attribute. If an attribute with that name is already present in the element, its value is changed to be that of the value parameter. This value is a simple string, it is not parsed as it is being set. So any markup (such as syntax to be recognized as an entity reference) is treated as literal text, and needs to be appropriately escaped by the implementation when it is written out. In order to assign an attribute value that contains entity references, the user must create an `Attr` [p.37] node plus any `Text` [p.42] and `EntityReference` [p.46] nodes, build the appropriate subtree, and use `setAttributeNode` to assign it as the value of an attribute.

Parameters

`name`	The name of the attribute to create or alter.
`value`	Value to set in string form.

Exceptions

`DOMException` [p.19]

INVALID_CHARACTER_ERR: Raised if the specified name contains an invalid character.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

`removeAttribute`

Removes an attribute by name. If the removed attribute has a default value it is immediately replaced.

Parameters

`name`	The name of the attribute to remove.

Exceptions

`DOMException` [p.19]

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

This method returns nothing.

`getAttributeNode`

Retrieves an `Attr` [p.37] node by name.

Parameters

`name`	The name of the attribute to retrieve.

Return Value

The `Attr` [p.37] node with the specified attribute name or `null` if there is no such attribute.

This method raises no exceptions.

`setAttributeNode`

Adds a new attribute. If an attribute with that name is already present in the element, it is replaced by the new one.

Parameters

`newAttr`	The `Attr` [p.37] node to add to the attribute list.

Return Value

If the `newAttr` attribute replaces an existing attribute with the same name, the previously existing `Attr` [p.37] node is returned, otherwise `null` is returned.

Exceptions

`DOMException` [p.19]

WRONG_DOCUMENT_ERR: Raised if `newAttr` was created from a different document than the one that created the element.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

INUSE_ATTRIBUTE_ERR: Raised if `newAttr` is already an attribute of another `Element` object. The DOM user must explicitly clone `Attr` [p.37] nodes to re-use them in other elements.

`removeAttributeNode`

Removes the specified attribute.

Parameters

`oldAttr`	The `Attr` [p.37] node to remove from the attribute list. If the removed `Attr` [p.37] has a default value it is immediately replaced.

Return Value

The `Attr` [p.37] node that was removed.

Exceptions

`DOMException` [p.19]

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

NOT_FOUND_ERR: Raised if `oldAttr` is not an attribute of the element.

`getElementsByTagName`

Returns a `NodeList` [p.32] of all descendant elements with a given tag name, in the order in which they would be encountered in a preorder traversal of the `Element` tree.

Parameters

`name`	The name of the tag to match on. The special value "*" matches all tags.

Return Value

A list of matching `Element` nodes.

This method raises no exceptions.

`normalize`

Puts all `Text` [p.42] nodes in the full depth of the sub-tree underneath this `Element` into a "normal" form where only markup (e.g., tags, comments, processing instructions, CDATA sections, and entity references) separates `Text` [p.42] nodes, i.e., there are no adjacent `Text` [p.42] nodes. This can be used to ensure that the DOM view of a document is the same as if it were saved and re-loaded, and is useful when operations (such as XPointer lookups) that depend on a particular document tree structure are to be used.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

Interface *Text*

The `Text` interface represents the textual content (termed character data in XML) of an `Element` [p.38] or `Attr` [p.37] . If there is no markup inside an element's content, the text is contained in a single object implementing the `Text` interface that is the only child of the element. If there is markup, it is parsed into a list of elements and `Text` nodes that form the list of children of the element.

When a document is first made available via the DOM, there is only one `Text` node for each block of text. Users may create adjacent `Text` nodes that represent the contents of a given element without any intervening markup, but should be aware that there is no way to represent the separations between these nodes in XML or HTML, so they will not (in general) persist between DOM editing sessions. The `normalize()` method on `Element` [p.38] merges any such adjacent `Text` objects into a single node for each block of text; this is recommended before employing operations that depend on a particular document structure, such as navigation with `XPointers` .

IDL Definition

```
interface Text : CharacterData {
  Text                      splitText(in unsigned long offset)
                                      raises(DOMException);
};
```

Methods

`splitText`

Breaks this `Text` node into two Text nodes at the specified offset, keeping both in the tree as siblings. This node then only contains all the content up to the `offset` point. And a new `Text` node, which is inserted as the next sibling of this node, contains all the content at and after the `offset` point.

Parameters

`offset`	The offset at which to split, starting from 0.

Return Value

The new `Text` node.

Exceptions

`DOMException` [p.19]

INDEX_SIZE_ERR: Raised if the specified offset is negative or greater than the number of characters in `data`.

NO_MODIFICATION_ALLOWED_ERR: Raised if this node is readonly.

Interface *Comment*

This represents the content of a comment, i.e., all the characters between the starting '`<!--`' and ending '`-->`'. Note that this is the definition of a comment in XML, and, in practice, HTML, although some HTML tools may implement the full SGML comment structure.

IDL Definition

```
interface Comment : CharacterData {
};
```

1.3. Extended Interfaces

The interfaces defined here form part of the DOM Level 1 Core specification, but objects that expose these interfaces will never be encountered in a DOM implementation that deals only with HTML. As such, HTML-only DOM implementations do not need to have objects that implement these interfaces.

Interface *CDATASection*

CDATA sections are used to escape blocks of text containing characters that would otherwise be regarded as markup. The only delimiter that is recognized in a CDATA section is the "]]>" string that ends the CDATA section. CDATA sections can not be nested. The primary purpose is for including material such as XML fragments, without needing to escape all the delimiters.

The `DOMString` attribute of the `Text` [p.42] node holds the text that is contained by the CDATA section. Note that this *may* contain characters that need to be escaped outside of CDATA sections and that, depending on the character encoding ("charset") chosen for serialization, it may be impossible to write out some characters as part of a CDATA section.

The `CDATASection` interface inherits the `CharacterData` [p.34] interface through the `Text` [p.42] interface. Adjacent `CDATASections` nodes are not merged by use of the Element.normalize() method.

IDL Definition

```
interface CDATASection : Text {
};
```

Interface *DocumentType*

Each Document [p.22] has a doctype attribute whose value is either null or a DocumentType object. The DocumentType interface in the DOM Level 1 Core provides an interface to the list of entities that are defined for the document, and little else because the effect of namespaces and the various XML scheme efforts on DTD representation are not clearly understood as of this writing.

The DOM Level 1 doesn't support editing DocumentType nodes.

IDL Definition

```
interface DocumentType : Node {
  readonly attribute  DOMString           name;
  readonly attribute  NamedNodeMap        entities;
  readonly attribute  NamedNodeMap        notations;
};
```

Attributes

name
: The name of DTD; i.e., the name immediately following the DOCTYPE keyword.

entities
: A NamedNodeMap [p.32] containing the general entities, both external and internal, declared in the DTD. Duplicates are discarded. For example in:

```
<!DOCTYPE ex SYSTEM "ex.dtd" [
  <!ENTITY foo "foo">
  <!ENTITY bar "bar">
  <!ENTITY % baz "baz">
]>
<ex/>
```

the interface provides access to foo and bar but not baz. Every node in this map also implements the Entity [p.45] interface.

The DOM Level 1 does not support editing entities, therefore entities cannot be altered in any way.

notations
: A NamedNodeMap [p.32] containing the notations declared in the DTD. Duplicates are discarded. Every node in this map also implements the Notation [p.44] interface.

The DOM Level 1 does not support editing notations, therefore notations cannot be altered in any way.

Interface *Notation*

This interface represents a notation declared in the DTD. A notation either declares, by name, the format of an unparsed entity (see section 4.7 of the XML 1.0 specification), or is used for formal declaration of Processing Instruction targets (see section 2.6 of the XML 1.0 specification). The `nodeName` attribute inherited from `Node` [p.25] is set to the declared name of the notation.

The DOM Level 1 does not support editing `Notation` nodes; they are therefore readonly.

A `Notation` node does not have any parent.

IDL Definition

```
interface Notation : Node {
  readonly attribute  DOMString            publicId;
  readonly attribute  DOMString            systemId;
};
```

Attributes

`publicId`
: The public identifier of this notation. If the public identifier was not specified, this is `null`.

`systemId`
: The system identifier of this notation. If the system identifier was not specified, this is `null`.

Interface *Entity*

This interface represents an entity, either parsed or unparsed, in an XML document. Note that this models the entity itself *not* the entity declaration. `Entity` declaration modeling has been left for a later Level of the DOM specification.

The `nodeName` attribute that is inherited from `Node` [p.25] contains the name of the entity.

An XML processor may choose to completely expand entities before the structure model is passed to the DOM; in this case there will be no `EntityReference` [p.46] nodes in the document tree.

XML does not mandate that a non-validating XML processor read and process entity declarations made in the external subset or declared in external parameter entities. This means that parsed entities declared in the external subset need not be expanded by some classes of applications, and that the replacement value of the entity may not be available. When the replacement value is available, the corresponding `Entity` node's child list represents the structure of that replacement text. Otherwise, the child list is empty.

The resolution of the children of the `Entity` (the replacement value) may be lazily evaluated; actions by the user (such as calling the `childNodes` method on the `Entity` Node) are assumed to trigger the evaluation.

The DOM Level 1 does not support editing `Entity` nodes; if a user wants to make changes to the contents of an `Entity`, every related `EntityReference` [p.46] node has to be replaced in the structure model by a clone of the `Entity`'s contents, and then the desired changes must be made to each of those clones instead. All the descendants of an `Entity` node are readonly.

An `Entity` node does not have any parent.

IDL Definition

```
interface Entity : Node {
  readonly attribute  DOMString           publicId;
  readonly attribute  DOMString           systemId;
  readonly attribute  DOMString           notationName;
};
```

Attributes

`publicId`
: The public identifier associated with the entity, if specified. If the public identifier was not specified, this is `null`.

`systemId`
: The system identifier associated with the entity, if specified. If the system identifier was not specified, this is `null`.

`notationName`
: For unparsed entities, the name of the notation for the entity. For parsed entities, this is `null`.

Interface *EntityReference*

`EntityReference` objects may be inserted into the structure model when an entity reference is in the source document, or when the user wishes to insert an entity reference. Note that character references and references to predefined entities are considered to be expanded by the HTML or XML processor so that characters are represented by their Unicode equivalent rather than by an entity reference. Moreover, the XML processor may completely expand references to entities while building the structure model, instead of providing `EntityReference` objects. If it does provide such objects, then for a given `EntityReference` node, it may be that there is no `Entity` [p.45] node representing the referenced entity; but if such an `Entity` [p.45] exists, then the child list of the `EntityReference` node is the same as that of the `Entity` [p.45] node. As with the `Entity` [p.45] node, all descendants of the `EntityReference` are readonly.

The resolution of the children of the `EntityReference` (the replacement value of the referenced `Entity` [p.45]) may be lazily evaluated; actions by the user (such as calling the `childNodes` method on the `EntityReference` node) are assumed to trigger the evaluation.

IDL Definition

```
interface EntityReference : Node {
};
```

Interface *ProcessingInstruction*

The `ProcessingInstruction` interface represents a "processing instruction", used in XML as a way to keep processor-specific information in the text of the document.

IDL Definition

```
interface ProcessingInstruction : Node {
  readonly attribute  DOMString           target;
           attribute  DOMString           data;
                                       // raises(DOMException) on setting
};
```

Attributes

`target`

The target of this processing instruction. XML defines this as being the first token following the markup that begins the processing instruction.

`data`

The content of this processing instruction. This is from the first non white space character after the target to the character immediately preceding the `?>`.

Exceptions on setting

`DOMException` [p.19]

NO_MODIFICATION_ALLOWED_ERR: Raised when the node is readonly.

2. Document Object Model (HTML) Level 1

Editors

Mike Champion, ArborText
Vidur Apparao, Netscape
Scott Isaacs, Microsoft (until January 1998)
Chris Wilson, Microsoft (after January 1998)
Ian Jacobs, W3C

2.1. Introduction

This section extends the Level 1 Core API to describe objects and methods specific to HTML documents. In general, the functionality needed to manipulate hierarchical document structures, elements, and attributes will be found in the core section; functionality that depends on the specific elements defined in HTML will be found in this section.

The goals of the HTML-specific DOM API are:

- to specialize and add functionality that relates specifically to HTML documents and elements.
- to address issues of backwards compatibility with the "DOM Level 0".
- to provide convenience mechanisms, where appropriate, for common and frequent operations on HTML documents.

The term "DOM Level 0" refers to a mix (not formally specified) of HTML document functionalities offered by Netscape Navigator version 3.0 and Microsoft Internet Explorer version 3.0. In some cases, attributes or methods have been included for reasons of backward compatibility with "DOM Level 0".

The key differences between the core DOM and the HTML application of DOM is that the HTML Document Object Model exposes a number of convenience methods and properties that are consistent with the existing models and are more appropriate to script writers. In many cases, these enhancements are not applicable to a general DOM because they rely on the presence of a predefined DTD. For DOM Level 1, the transitional and frameset DTDs for HTML 4.0 are assumed. Interoperability between implementations is only guaranteed for elements and attributes that are specified in these DTDs.

More specifically, this document includes the following specializations for HTML:

- An HTMLDocument interface, derived from the core Document interface. HTMLDocument specifies the operations and queries that can be made on a HTML document.
- An HTMLElement interface, derived from the core Element interface. HTMLElement specifies the operations and queries that can be made on any HTML element. Methods on HTMLElement include those that allow for the retrieval and modification of attributes that apply to all HTML elements.
- Specializations for all HTML elements that have attributes that extend beyond those specified in the HTMLElement interface. For all such attributes, the derived interface for the element contains explicit methods for setting and getting the values.

The DOM Level 1 does not include mechanisms to access and modify style specified through CSS 1. Furthermore, it does not define an event model for HTML documents. This functionality is planned to be specified in a future Level of this specification.

2.2. HTML Application of Core DOM

2.2.1. Naming Conventions

The HTML DOM follows a naming convention for properties, methods, events, collections, and data types. All names are defined as one or more English words concatenated together to form a single string. Properties and Methods

The property or method name starts with the initial keyword in lowercase, and each subsequent word starts with a capital letter. For example, a property that returns document meta information such as the date the file was created might be named "fileDateCreated". In the ECMAScript binding, properties are exposed as properties of a given object. In Java, properties are exposed with get and set methods. Non-HTML 4.0 interfaces and attributes

While most of the interfaces defined below can be mapped directly to elements defined in the HTML 4.0 Recommendation, some of them cannot. Similarly, not all attributes listed below have counterparts in the HTML 4.0 specification (and some do, but have been renamed to avoid conflicts with scripting languages). Interfaces and attribute definitions that have links to the HTML 4.0 specification have corresponding element and attribute definitions there; all others are added by this specification, either for convenience or backwards compatibility with "DOM Level 0" implementations.

2.3. Miscellaneous Object Definitions

Interface *HTMLCollection*

An `HTMLCollection` is a list of nodes. An individual node may be accessed by either ordinal index or the node's `name` or `id` attributes. *Note:* Collections in the HTML DOM are assumed to be *live* meaning that they are automatically updated when the underlying document is changed.

IDL Definition

```
interface HTMLCollection {
  readonly attribute  unsigned long       length;
  Node                          item(in unsigned long index);
  Node                          namedItem(in DOMString name);
};
```

Attributes

`length`

This attribute specifies the length or *size* of the list.

Methods

`item`

This method retrieves a node specified by ordinal index. Nodes are numbered in tree order (depth-first traversal order).

Parameters

`index`	The index of the node to be fetched. The index origin is 0.

Return Value

The `Node` [p.25] at the corresponding position upon success. A value of `null` is returned if the index is out of range.

This method raises no exceptions.

`namedItem`

This method retrieves a `Node` [p.25] using a name. It first searches for a `Node` [p.25] with a matching `id` attribute. If it doesn't find one, it then searches for a `Node` [p.25] with a matching `name` attribute, but only on those elements that are allowed a name attribute.

Parameters

`name`	The name of the `Node` [p.25] to be fetched.

Return Value

The `Node` [p.25] with a `name` or `id` attribute whose value corresponds to the specified string. Upon failure (e.g., no node with this name exists), returns `null`.

This method raises no exceptions.

2.4. Objects related to HTML documents

Interface *HTMLDocument*

An `HTMLDocument` is the root of the HTML hierarchy and holds the entire content. Beside providing access to the hierarchy, it also provides some convenience methods for accessing certain sets of information from the document.

The following properties have been deprecated in favor of the corresponding ones for the BODY element:

- alinkColor
- background
- bgColor
- fgColor
- linkColor
- vlinkColor

IDL Definition

```
interface HTMLDocument : Document {
           attribute  DOMString          title;
  readonly attribute  DOMString          referrer;
  readonly attribute  DOMString          domain;
  readonly attribute  DOMString          URL;
           attribute  HTMLElement        body;
  readonly attribute  HTMLCollection     images;
  readonly attribute  HTMLCollection     applets;
  readonly attribute  HTMLCollection     links;
  readonly attribute  HTMLCollection     forms;
  readonly attribute  HTMLCollection     anchors;
           attribute  DOMString          cookie;
```

```
  void                         open();
  void                         close();
  void                         write(in DOMString text);
  void                         writeln(in DOMString text);
  Element                      getElementById(in DOMString elementId);
  NodeList                     getElementsByName(in DOMString elementName);
};
```

Attributes

`title`
: The title of a document as specified by the `TITLE` element in the head of the document.

`referrer`
: Returns the URI of the page that linked to this page. The value is an empty string if the user navigated to the page directly (not through a link, but, for example, via a bookmark).

`domain`
: The domain name of the server that served the document, or a null string if the server cannot be identified by a domain name.

`URL`
: The complete URI of the document.

`body`
: The element that contains the content for the document. In documents with `BODY` contents, returns the `BODY` element, and in frameset documents, this returns the outermost `FRAMESET` element.

`images`
: A collection of all the `IMG` elements in a document. The behavior is limited to `IMG` elements for backwards compatibility.

`applets`
: A collection of all the `OBJECT` elements that include applets and `APPLET` (*deprecated*) elements in a document.

`links`
: A collection of all `AREA` elements and anchor (`A`) elements in a document with a value for the `href` attribute.

`forms`
: A collection of all the forms of a document.

`anchors`
: A collection of all the anchor (`A`) elements in a document with a value for the `name` attribute.*Note.* For reasons of backwards compatibility, the returned set of anchors only contains those anchors created with the `name` attribute, not those created with the `id` attribute.

`cookie`
: The cookies associated with this document. If there are none, the value is an empty string. Otherwise, the value is a string: a semicolon-delimited list of "name, value" pairs for all the cookies associated with the page. For example, `name=value;expires=date`.

Methods

`open`
: *Note.* This method and the ones following allow a user to add to or replace the structure model of a document using strings of unparsed HTML. At the time of writing alternate methods for providing similar functionality for both HTML and XML documents were

being considered. The following methods may be deprecated at some point in the future in favor of a more general-purpose mechanism.

Open a document stream for writing. If a document exists in the target, this method clears it.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

`close`

Closes a document stream opened by `open()` and forces rendering.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

`write`

Write a string of text to a document stream opened by `open()`. The text is parsed into the document's structure model.

Parameters

`text`	The string to be parsed into some structure in the document structure model.

This method returns nothing.
This method raises no exceptions.

`writeln`

Write a string of text followed by a newline character to a document stream opened by `open()`. The text is parsed into the document's structure model.

Parameters

`text`	The string to be parsed into some structure in the document structure model.

This method returns nothing.
This method raises no exceptions.

`getElementById`

Returns the Element whose `id` is given by elementId. If no such element exists, returns `null`. Behavior is not defined if more than one element has this `id`.

Parameters

`elementId`	The unique `id` value for an element.

Return Value

The matching element.

This method raises no exceptions.

`getElementsByName`

Returns the (possibly empty) collection of elements whose `name` value is given by `elementName`.

Parameters

`elementName`	The `name` attribute value for an element.

Return Value

The matching elements.

This method raises no exceptions.

2.5. HTML Elements

2.5.1. Property Attributes

HTML attributes are exposed as properties on the element object. The name of the exposed property always uses the naming conventions, and is independent of the case of the attribute in the source document. The data type of the property is determined by the type of the attribute as determined by the HTML 4.0 transitional and frameset DTDs. The attributes have the semantics (including case-sensitivity) given in the HTML 4.0 specification.

The attributes are exposed as properties for compatibility with "DOM Level 0". This usage is deprecated because it can not be generalized to all possible attribute names, as is required both for XML and potentially for future versions of HTML. We recommend the use of generic methods on the core Element interface for setting, getting and removing attributes.

DTD Data Type	Object Model Data Type
CDATA	DOMString
Value list (e.g., (left | right | center))	DOMString
one-value Value list (e.g., (border))	boolean
Number	long int

The return value of an attribute that has a data type that is a value list is always capitalized, independent of the case of the value in the source document. For example, if the value of the align attribute on a P element is "left" then it is returned as "Left". For attributes with the `CDATA` data type, the case of the return value is that given in the source document.

2.5.2. Naming Exceptions

To avoid name-space conflicts, an attribute with the same name as a keyword in one of our chosen binding languages is prefixed. For HTML, the prefix used is "html". For example, the `for` attribute of the `LABEL` element collides with loop construct naming conventions and is renamed `htmlFor`.

2.5.3. Exposing Element Type Names (tagName)

The element type names exposed through a property are in uppercase. For example, the body element type name is exposed through the "tagName" property as "BODY".

2.5.4. The HTMLElement interface

Interface *HTMLElement*

All HTML element interfaces derive from this class. Elements that only expose the HTML core attributes are represented by the base `HTMLElement` interface. These elements are as follows:

- HEAD
- special: SUB, SUP, SPAN, BDO
- font: TT, I, B, U, S, STRIKE, BIG, SMALL
- phrase: EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE, ACRONYM, ABBR
- list: DD, DT
- NOFRAMES, NOSCRIPT
- ADDRESS, CENTER

Note. The `style` attribute for this interface is reserved for future usage.

IDL Definition

```
interface HTMLElement : Element {
           attribute  DOMString            id;
           attribute  DOMString            title;
           attribute  DOMString            lang;
           attribute  DOMString            dir;
           attribute  DOMString            className;
};
```

Attributes

`id`
: The element's identifier. See the id attribute definition in HTML 4.0.

`title`
: The element's advisory title. See the title attribute definition in HTML 4.0.

`lang`
: Language code defined in RFC 1766. See the lang attribute definition in HTML 4.0.

`dir`
: Specifies the base direction of directionally neutral text and the directionality of tables. See the dir attribute definition in HTML 4.0.

`className`
The class attribute of the element. This attribute has been renamed due to conflicts with the "class" keyword exposed by many languages. See the class attribute definition in HTML 4.0.

2.5.5. Object definitions

Interface *HTMLHtmlElement*

Root of an HTML document. See the HTML element definition in HTML 4.0.
IDL Definition

```
interface HTMLHtmlElement : HTMLElement {
           attribute  DOMString           version;
};
```

Attributes
`version`
Version information about the document's DTD. See the version attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLHeadElement*

Document head information. See the HEAD element definition in HTML 4.0.
IDL Definition

```
interface HTMLHeadElement : HTMLElement {
           attribute  DOMString           profile;
};
```

Attributes
`profile`
URI designating a metadata profile. See the profile attribute definition in HTML 4.0.

Interface *HTMLLinkElement*

The `LINK` element specifies a link to an external resource, and defines this document's relationship to that resource (or vice versa). See the LINK element definition in HTML 4.0.
IDL Definition

```
interface HTMLLinkElement : HTMLElement {
           attribute  boolean             disabled;
           attribute  DOMString           charset;
           attribute  DOMString           href;
           attribute  DOMString           hreflang;
           attribute  DOMString           media;
           attribute  DOMString           rel;
           attribute  DOMString           rev;
           attribute  DOMString           target;
           attribute  DOMString           type;
};
```

Attributes

`disabled`
: Enables/disables the link. This is currently only used for style sheet links, and may be used to activate or deactivate style sheets.

`charset`
: The character encoding of the resource being linked to. See the charset attribute definition in HTML 4.0.

`href`
: The URI of the linked resource. See the href attribute definition in HTML 4.0.

`hreflang`
: Language code of the linked resource. See the hreflang attribute definition in HTML 4.0.

`media`
: Designed for use with one or more target media. See the media attribute definition in HTML 4.0.

`rel`
: Forward link type. See the rel attribute definition in HTML 4.0.

`rev`
: Reverse link type. See the rev attribute definition in HTML 4.0.

`target`
: Frame to render the resource in. See the target attribute definition in HTML 4.0.

`type`
: Advisory content type. See the type attribute definition in HTML 4.0.

Interface *HTMLTitleElement*

The document title. See the TITLE element definition in HTML 4.0.

IDL Definition

```
interface HTMLTitleElement : HTMLElement {
          attribute  DOMString            text;
};
```

Attributes

`text`
: The specified title as a string.

Interface *HTMLMetaElement*

This contains generic meta-information about the document. See the META element definition in HTML 4.0.

IDL Definition

```
interface HTMLMetaElement : HTMLElement {
          attribute  DOMString            content;
          attribute  DOMString            httpEquiv;
          attribute  DOMString            name;
          attribute  DOMString            scheme;
};
```

Attributes

`content`
: Associated information. See the content attribute definition in HTML 4.0.

`httpEquiv`
: HTTP response header name. See the http-equiv attribute definition in HTML 4.0.

`name`
: Meta information name. See the name attribute definition in HTML 4.0.

`scheme`
: Select form of content. See the scheme attribute definition in HTML 4.0.

Interface *HTMLBaseElement*

Document base URI. See the BASE element definition in HTML 4.0.

IDL Definition

```
interface HTMLBaseElement : HTMLElement {
           attribute  DOMString              href;
           attribute  DOMString              target;
};
```

Attributes

`href`
: The base URI See the href attribute definition in HTML 4.0.

`target`
: The default target frame. See the target attribute definition in HTML 4.0.

Interface *HTMLIsIndexElement*

This element is used for single-line text input. See the ISINDEX element definition in HTML 4.0. This element is deprecated in HTML 4.0.

IDL Definition

```
interface HTMLIsIndexElement : HTMLElement {
  readonly attribute  HTMLFormElement        form;
           attribute  DOMString              prompt;
};
```

Attributes

`form`
: Returns the FORM element containing this control. Returns null if this control is not within the context of a form.

`prompt`
: The prompt message. See the prompt attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLStyleElement*

Style information. A more detailed style sheet object model is planned to be defined in a separate document. See the STYLE element definition in HTML 4.0.

IDL Definition

```
interface HTMLStyleElement : HTMLElement {
          attribute  boolean              disabled;
          attribute  DOMString            media;
          attribute  DOMString            type;
};
```

Attributes

`disabled`
: Enables/disables the style sheet.

`media`
: Designed for use with one or more target media. See the media attribute definition in HTML 4.0.

`type`
: The style sheet language (Internet media type). See the type attribute definition in HTML 4.0.

Interface *HTMLBodyElement*

The HTML document body. This element is always present in the DOM API, even if the tags are not present in the source document. See the BODY element definition in HTML 4.0.

IDL Definition

```
interface HTMLBodyElement : HTMLElement {
          attribute  DOMString            aLink;
          attribute  DOMString            background;
          attribute  DOMString            bgColor;
          attribute  DOMString            link;
          attribute  DOMString            text;
          attribute  DOMString            vLink;
};
```

Attributes

`aLink`
: Color of active links (after mouse-button down, but before mouse-button up). See the alink attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`background`
: URI of the background texture tile image. See the background attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`bgColor`
: Document background color. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`link`
: Color of links that are not active and unvisited. See the link attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`text`
: Document text color. See the text attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`vLink`
: Color of links that have been visited by the user. See the vlink attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLFormElement*

The `FORM` element encompasses behavior similar to a collection and an element. It provides direct access to the contained input elements as well as the attributes of the form element. See the FORM element definition in HTML 4.0.

IDL Definition

```
interface HTMLFormElement : HTMLElement {
  readonly attribute  HTMLCollection       elements;
  readonly attribute  long                 length;
           attribute  DOMString            name;
           attribute  DOMString            acceptCharset;
           attribute  DOMString            action;
           attribute  DOMString            enctype;
           attribute  DOMString            method;
           attribute  DOMString            target;
  void                      submit();
  void                      reset();
};
```

Attributes

`elements`
: Returns a collection of all control elements in the form.

`length`
: The number of form controls in the form.

`name`
: Names the form.

`acceptCharset`
: List of character sets supported by the server. See the accept-charset attribute definition in HTML 4.0.

`action`
: Server-side form handler. See the action attribute definition in HTML 4.0.

`enctype`
: The content type of the submitted form, generally "application/x-www-form-urlencoded". See the enctype attribute definition in HTML 4.0.

`method`
: HTTP method used to submit form. See the method attribute definition in HTML 4.0.

`target`
: Frame to render the resource in. See the target attribute definition in HTML 4.0.

Methods

`submit`
: Submits the form. It performs the same action as a submit button.
: This method has no parameters.
: This method returns nothing.
: This method raises no exceptions.

`reset`
Restores a form element's default values. It performs the same action as a reset button.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

Interface *HTMLSelectElement*

The select element allows the selection of an option. The contained options can be directly accessed through the select element as a collection. See the SELECT element definition in HTML 4.0.

IDL Definition

```
interface HTMLSelectElement : HTMLElement {
  readonly attribute  DOMString           type;
           attribute  long                selectedIndex;
           attribute  DOMString           value;
  readonly attribute  long                length;
  readonly attribute  HTMLFormElement     form;
  readonly attribute  HTMLCollection      options;
           attribute  boolean             disabled;
           attribute  boolean             multiple;
           attribute  DOMString           name;
           attribute  long                size;
           attribute  long                tabIndex;
  void                      add(in HTMLElement element,
                                in HTMLElement before);
  void                      remove(in long index);
  void                      blur();
  void                      focus();
};
```

Attributes

`type`
The type of control created.

`selectedIndex`
The ordinal index of the selected option. The value -1 is returned if no element is selected. If multiple options are selected, the index of the first selected option is returned.

`value`
The current form control value.

`length`
The number of options in this `SELECT`.

`form`
Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

`options`
The collection of `OPTION` elements contained by this element.

`disabled`
The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

`multiple`
: If true, multiple `OPTION` elements may be selected in this `SELECT`. See the multiple attribute definition in HTML 4.0.

`name`
: Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`size`
: Number of visible rows. See the size attribute definition in HTML 4.0.

`tabIndex`
: Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

Methods

`add`
: Add a new element to the collection of `OPTION` elements for this `SELECT`.

Parameters

`element`	The element to add.
`before`	The element to insert before, or NULL for the head of the list.

This method returns nothing.
This method raises no exceptions.

`remove`
: Remove an element from the collection of `OPTION` elements for this `SELECT`. Does nothing if no element has the given index.

Parameters

`index`	The index of the item to remove.

This method returns nothing.
This method raises no exceptions.

`blur`
: Removes keyboard focus from this element.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

`focus`
: Gives keyboard focus to this element.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

Interface *HTMLOptGroupElement*

Group options together in logical subdivisions. See the OPTGROUP element definition in HTML 4.0.

IDL Definition

```
interface HTMLOptGroupElement : HTMLElement {
          attribute  boolean              disabled;
          attribute  DOMString            label;
};
```

Attributes

`disabled`
: The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

`label`
: Assigns a label to this option group. See the label attribute definition in HTML 4.0.

Interface *HTMLOptionElement*

A selectable choice. See the OPTION element definition in HTML 4.0.

IDL Definition

```
interface HTMLOptionElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  boolean              defaultSelected;
  readonly attribute  DOMString            text;
           attribute  long                 index;
           attribute  boolean              disabled;
           attribute  DOMString            label;
  readonly attribute  boolean              selected;
           attribute  DOMString            value;
};
```

Attributes

`form`
: Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

`defaultSelected`
: Stores the initial value of the `selected` attribute.

`text`
: The text contained within the option element.

`index`
: The index of this `OPTION` in its parent `SELECT`.

`disabled`
: The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

`label`
: Option label for use in hierarchical menus. See the label attribute definition in HTML 4.0.

`selected`
: Means that this option is initially selected. See the selected attribute definition in HTML 4.0.

`value`
: The current form control value. See the value attribute definition in HTML 4.0.

Interface *HTMLInputElement*

Form control. *Note.* Depending upon the environment the page is being viewed, the value property may be read-only for the file upload input type. For the "password" input type, the actual value returned may be masked to prevent unauthorized use. See the INPUT element definition in HTML 4.0.

IDL Definition

```
interface HTMLInputElement : HTMLElement {
           attribute  DOMString             defaultValue;
           attribute  boolean               defaultChecked;
  readonly attribute  HTMLFormElement       form;
           attribute  DOMString             accept;
           attribute  DOMString             accessKey;
           attribute  DOMString             align;
           attribute  DOMString             alt;
           attribute  boolean               checked;
           attribute  boolean               disabled;
           attribute  long                  maxLength;
           attribute  DOMString             name;
           attribute  boolean               readOnly;
           attribute  DOMString             size;
           attribute  DOMString             src;
           attribute  long                  tabIndex;
  readonly attribute  DOMString             type;
           attribute  DOMString             useMap;
           attribute  DOMString             value;
  void                      blur();
  void                      focus();
  void                      select();
  void                      click();
};
```

Attributes

`defaultValue`
: Stores the initial control value (i.e., the initial value of `value`).

`defaultChecked`
: When `type` has the value "Radio" or "Checkbox", stores the initial value of the `checked` attribute.

`form`
: Returns the FORM element containing this control. Returns null if this control is not within the context of a form.

`accept`
: A comma-separated list of content types that a server processing this form will handle correctly. See the accept attribute definition in HTML 4.0.

`accessKey`
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`align`
: Aligns this object (vertically or horizontally) with respect to its surrounding text. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`alt`
: Alternate text for user agents not rendering the normal content of this element. See the alt attribute definition in HTML 4.0.

`checked`
: Describes whether a radio or check box is checked, when `type` has the value "Radio" or "Checkbox". The value is TRUE if explicitly set. Represents the current state of the checkbox or radio button. See the checked attribute definition in HTML 4.0.

`disabled`
: The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

`maxLength`
: Maximum number of characters for text fields, when `type` has the value "Text" or "Password". See the maxlength attribute definition in HTML 4.0.

`name`
: Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`readOnly`
: This control is read-only. When `type` has the value "text" or "password" only. See the readonly attribute definition in HTML 4.0.

`size`
: Size information. The precise meaning is specific to each type of field. See the size attribute definition in HTML 4.0.

`src`
: When the `type` attribute has the value "Image", this attribute specifies the location of the image to be used to decorate the graphical submit button. See the src attribute definition in HTML 4.0.

`tabIndex`
: Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`type`
: The type of control created. See the type attribute definition in HTML 4.0.

`useMap`
: Use client-side image map. See the usemap attribute definition in HTML 4.0.

`value`
: The current form control value. Used for radio buttons and check boxes. See the value attribute definition in HTML 4.0.

Methods

`blur`
: Removes keyboard focus from this element.
: This method has no parameters.
: This method returns nothing.
: This method raises no exceptions.

`focus`

Gives keyboard focus to this element.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

`select`

Select the contents of the text area. For `INPUT` elements whose `type` attribute has one of the following values: "Text", "File", or "Password".
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

`click`

Simulate a mouse-click. For `INPUT` elements whose `type` attribute has one of the following values: "Button", "Checkbox", "Radio", "Reset", or "Submit".
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

Interface *HTMLTextAreaElement*

Multi-line text field. See the TEXTAREA element definition in HTML 4.0.

IDL Definition

```
interface HTMLTextAreaElement : HTMLElement {
           attribute  DOMString              defaultValue;
  readonly attribute  HTMLFormElement        form;
           attribute  DOMString              accessKey;
           attribute  long                   cols;
           attribute  boolean                disabled;
           attribute  DOMString              name;
           attribute  boolean                readOnly;
           attribute  long                   rows;
           attribute  long                   tabIndex;
  readonly attribute  DOMString              type;
           attribute  DOMString              value;
  void                      blur();
  void                      focus();
  void                      select();
};
```

Attributes

`defaultValue`

Stores the initial control value (i.e., the initial value of `value`).

`form`

Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

`accessKey`

A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`cols`
  Width of control (in characters). See the cols attribute definition in HTML 4.0.
`disabled`
  The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.
`name`
  Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.
`readOnly`
  This control is read-only. See the readonly attribute definition in HTML 4.0.
`rows`
  Number of text rows. See the rows attribute definition in HTML 4.0.
`tabIndex`
  Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.
`type`
  The type of this form control.
`value`
  The current textual content of the multi-line text field. If the entirety of the data can not fit into a single wstring, the implementation may truncate the data.

Methods

`blur`
  Removes keyboard focus from this element.
  This method has no parameters.
  This method returns nothing.
  This method raises no exceptions.
`focus`
  Gives keyboard focus to this element.
  This method has no parameters.
  This method returns nothing.
  This method raises no exceptions.
`select`
  Select the contents of the `TEXTAREA`.
  This method has no parameters.
  This method returns nothing.
  This method raises no exceptions.

Interface *HTMLButtonElement*

Push button. See the BUTTON element definition in HTML 4.0.

IDL Definition

```
interface HTMLButtonElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  DOMString            accessKey;
           attribute  boolean              disabled;
           attribute  DOMString            name;
           attribute  long                 tabIndex;
  readonly attribute  DOMString            type;
           attribute  DOMString            value;
};
```

Attributes

`form`
: Returns the FORM element containing this control. Returns null if this control is not within the context of a form.

`accessKey`
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`disabled`
: The control is unavailable in this context. See the disabled attribute definition in HTML 4.0.

`name`
: Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`tabIndex`
: Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`type`
: The type of button. See the type attribute definition in HTML 4.0.

`value`
: The current form control value. See the value attribute definition in HTML 4.0.

Interface *HTMLLabelElement*

Form field label text. See the LABEL element definition in HTML 4.0.

IDL Definition

```
interface HTMLLabelElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  DOMString            accessKey;
           attribute  DOMString            htmlFor;
};
```

Attributes

`form`
: Returns the FORM element containing this control. Returns null if this control is not within the context of a form.

`accessKey`
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`htmlFor`
: This attribute links this label with another form control by `id` attribute. See the for attribute definition in HTML 4.0.

Interface *HTMLFieldSetElement*

Organizes form controls into logical groups. See the FIELDSET element definition in HTML 4.0.

IDL Definition

```
interface HTMLFieldSetElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
};
```

Attributes

`form`
: Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

Interface *HTMLLegendElement*

Provides a caption for a `FIELDSET` grouping. See the LEGEND element definition in HTML 4.0.

IDL Definition

```
interface HTMLLegendElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  DOMString            accessKey;
           attribute  DOMString            align;
};
```

Attributes

`form`
: Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

`accessKey`
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`align`
: Text alignment relative to `FIELDSET`. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLUListElement*

Unordered list. See the UL element definition in HTML 4.0.

IDL Definition

```
interface HTMLUListElement : HTMLElement {
           attribute  boolean              compact;
           attribute  DOMString            type;
};
```

Attributes

`compact`
: Reduce spacing between list items. See the compact attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`type`
: Bullet style. See the type attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLOListElement*

Ordered list. See the OL element definition in HTML 4.0.

IDL Definition

```
interface HTMLOListElement : HTMLElement {
          attribute  boolean              compact;
          attribute  long                 start;
          attribute  DOMString            type;
};
```

Attributes

`compact`
: Reduce spacing between list items. See the compact attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`start`
: Starting sequence number. See the start attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`type`
: Numbering style. See the type attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLDListElement*

Definition list. See the DL element definition in HTML 4.0.

IDL Definition

```
interface HTMLDListElement : HTMLElement {
          attribute  boolean              compact;
};
```

Attributes

`compact`
: Reduce spacing between list items. See the compact attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLDirectoryElement*

Directory list. See the DIR element definition in HTML 4.0. This element is deprecated in HTML 4.0.

IDL Definition

```
interface HTMLDirectoryElement : HTMLElement {
          attribute  boolean               compact;
};
```

Attributes

`compact`
: Reduce spacing between list items. See the compact attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLMenuElement*

Menu list. See the MENU element definition in HTML 4.0. This element is deprecated in HTML 4.0.

IDL Definition

```
interface HTMLMenuElement : HTMLElement {
          attribute  boolean               compact;
};
```

Attributes

`compact`
: Reduce spacing between list items. See the compact attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLLIElement*

List item. See the LI element definition in HTML 4.0.

IDL Definition

```
interface HTMLLIElement : HTMLElement {
          attribute  DOMString             type;
          attribute  long                  value;
};
```

Attributes

`type`
: List item bullet style. See the type attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`value`
: Reset sequence number when used in `OL` See the value attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLBlockquoteElement*

??? See the BLOCKQUOTE element definition in HTML 4.0.

IDL Definition

```
interface HTMLBlockquoteElement : HTMLElement {
          attribute  DOMString             cite;
};
```

Attributes

`cite`

A URI designating a document that describes the reason for the change. See the cite attribute definition in HTML 4.0.

Interface *HTMLDivElement*

Generic block container. See the DIV element definition in HTML 4.0.

IDL Definition

```
interface HTMLDivElement : HTMLElement {
           attribute  DOMString            align;
};
```

Attributes

`align`

Horizontal text alignment. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLParagraphElement*

Paragraphs. See the P element definition in HTML 4.0.

IDL Definition

```
interface HTMLParagraphElement : HTMLElement {
           attribute  DOMString            align;
};
```

Attributes

`align`

Horizontal text alignment. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLHeadingElement*

For the `H1` to `H6` elements. See the H1 element definition in HTML 4.0.

IDL Definition

```
interface HTMLHeadingElement : HTMLElement {
           attribute  DOMString            align;
};
```

Attributes

`align`

Horizontal text alignment. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLQuoteElement*

For the Q and BLOCKQUOTE elements. See the Q element definition in HTML 4.0.
IDL Definition

```
interface HTMLQuoteElement : HTMLElement {
           attribute  DOMString            cite;
};
```

Attributes
`cite`
A URI designating a document that designates a source document or message. See the cite attribute definition in HTML 4.0.

Interface *HTMLPreElement*

Preformatted text. See the PRE element definition in HTML 4.0.
IDL Definition

```
interface HTMLPreElement : HTMLElement {
           attribute  long                 width;
};
```

Attributes
`width`
Fixed width for content. See the width attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLBRElement*

Force a line break. See the BR element definition in HTML 4.0.
IDL Definition

```
interface HTMLBRElement : HTMLElement {
           attribute  DOMString            clear;
};
```

Attributes
`clear`
Control flow of text around floats. See the clear attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLBaseFontElement*

Base font. See the BASEFONT element definition in HTML 4.0. This element is deprecated in HTML 4.0.
IDL Definition

```
interface HTMLBaseFontElement : HTMLElement {
           attribute  DOMString            color;
           attribute  DOMString            face;
           attribute  DOMString            size;
};
```

Attributes

`color`
: Font color. See the color attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`face`
: Font face identifier. See the face attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`size`
: Font size. See the size attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLFontElement*

Local change to font. See the FONT element definition in HTML 4.0. This element is deprecated in HTML 4.0.

IDL Definition

```
interface HTMLFontElement : HTMLElement {
          attribute  DOMString            color;
          attribute  DOMString            face;
          attribute  DOMString            size;
};
```

Attributes

`color`
: Font color. See the color attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`face`
: Font face identifier. See the face attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`size`
: Font size. See the size attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLHRElement*

Create a horizontal rule. See the HR element definition in HTML 4.0.

IDL Definition

```
interface HTMLHRElement : HTMLElement {
          attribute  DOMString            align;
          attribute  boolean              noShade;
          attribute  DOMString            size;
          attribute  DOMString            width;
};
```

Attributes

`align`
: Align the rule on the page. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`noShade`
: Indicates to the user agent that there should be no shading in the rendering of this element. See the noshade attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`size`
: The height of the rule. See the size attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`width`
: The width of the rule. See the width attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLModElement*

Notice of modification to part of a document. See the INS and DEL element definitions in HTML 4.0.

IDL Definition

```
interface HTMLModElement : HTMLElement {
           attribute  DOMString            cite;
           attribute  DOMString            dateTime;
};
```

Attributes

`cite`
: A URI designating a document that describes the reason for the change. See the cite attribute definition in HTML 4.0.

`dateTime`
: The date and time of the change. See the datetime attribute definition in HTML 4.0.

Interface *HTMLAnchorElement*

The anchor element. See the A element definition in HTML 4.0.

IDL Definition

```
interface HTMLAnchorElement : HTMLElement {
           attribute  DOMString            accessKey;
           attribute  DOMString            charset;
           attribute  DOMString            coords;
           attribute  DOMString            href;
           attribute  DOMString            hreflang;
           attribute  DOMString            name;
           attribute  DOMString            rel;
           attribute  DOMString            rev;
           attribute  DOMString            shape;
           attribute  long                 tabIndex;
           attribute  DOMString            target;
           attribute  DOMString            type;
  void                      blur();
  void                      focus();
};
```

Attributes

`accessKey`
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`charset`
: The character encoding of the linked resource. See the charset attribute definition in HTML 4.0.

`coords`
: Comma-separated list of lengths, defining an active region geometry. See also `shape` for the shape of the region. See the coords attribute definition in HTML 4.0.

`href`
: The URI of the linked resource. See the href attribute definition in HTML 4.0.

`hreflang`
: Language code of the linked resource. See the hreflang attribute definition in HTML 4.0.

`name`
: Anchor name. See the name attribute definition in HTML 4.0.

`rel`
: Forward link type. See the rel attribute definition in HTML 4.0.

`rev`
: Reverse link type. See the rev attribute definition in HTML 4.0.

`shape`
: The shape of the active area. The coordinates are given by `coords`. See the shape attribute definition in HTML 4.0.

`tabIndex`
: Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`target`
: Frame to render the resource in. See the target attribute definition in HTML 4.0.

`type`
: Advisory content type. See the type attribute definition in HTML 4.0.

Methods

`blur`
: Removes keyboard focus from this element.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

`focus`
: Gives keyboard focus to this element.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

Interface *HTMLImageElement*

Embedded image. See the IMG element definition in HTML 4.0.

IDL Definition

```
interface HTMLImageElement : HTMLElement {
           attribute  DOMString            lowSrc;
           attribute  DOMString            name;
           attribute  DOMString            align;
           attribute  DOMString            alt;
           attribute  DOMString            border;
           attribute  DOMString            height;
           attribute  DOMString            hspace;
           attribute  boolean              isMap;
           attribute  DOMString            longDesc;
           attribute  DOMString            src;
           attribute  DOMString            useMap;
           attribute  DOMString            vspace;
           attribute  DOMString            width;
};
```

Attributes

`lowSrc`
: URI designating the source of this image, for low-resolution output.

`name`
: The name of the element (for backwards compatibility).

`align`
: Aligns this object (vertically or horizontally) with respect to its surrounding text. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`alt`
: Alternate text for user agents not rendering the normal content of this element. See the alt attribute definition in HTML 4.0.

`border`
: Width of border around image. See the border attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`height`
: Override height. See the height attribute definition in HTML 4.0.

`hspace`
: Horizontal space to the left and right of this image. See the hspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`isMap`
: Use server-side image map. See the ismap attribute definition in HTML 4.0.

`longDesc`
: URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

`src`
: URI designating the source of this image. See the src attribute definition in HTML 4.0.

`useMap`
: Use client-side image map. See the usemap attribute definition in HTML 4.0.

`vspace`
: Vertical space above and below this image. See the vspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`width`
: Override width. See the width attribute definition in HTML 4.0.

Interface *HTMLObjectElement*

Generic embedded object. *Note.* In principle, all properties on the object element are read-write but in some environments some properties may be read-only once the underlying object is instantiated. See the OBJECT element definition in HTML 4.0.

IDL Definition

```
interface HTMLObjectElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  DOMString            code;
           attribute  DOMString            align;
           attribute  DOMString            archive;
           attribute  DOMString            border;
           attribute  DOMString            codeBase;
           attribute  DOMString            codeType;
           attribute  DOMString            data;
           attribute  boolean              declare;
           attribute  DOMString            height;
           attribute  DOMString            hspace;
           attribute  DOMString            name;
           attribute  DOMString            standby;
           attribute  long                 tabIndex;
           attribute  DOMString            type;
           attribute  DOMString            useMap;
           attribute  DOMString            vspace;
           attribute  DOMString            width;
};
```

Attributes

`form`
: Returns the `FORM` element containing this control. Returns null if this control is not within the context of a form.

`code`
: Applet class file. See the `code` attribute for HTMLAppletElement.

`align`
: Aligns this object (vertically or horizontally) with respect to its surrounding text. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`archive`
: Space-separated list of archives. See the archive attribute definition in HTML 4.0.

`border`
: Width of border around the object. See the border attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`codeBase`
: Base URI for `classid`, `data`, and `archive` attributes. See the codebase attribute definition in HTML 4.0.

`codeType`
: Content type for data downloaded via `classid` attribute. See the codetype attribute definition in HTML 4.0.

`data`
: A URI specifying the location of the object's data. See the data attribute definition in HTML 4.0.

`declare`
: Declare (for future reference), but do not instantiate, this object. See the declare attribute definition in HTML 4.0.

`height`
: Override height. See the height attribute definition in HTML 4.0.

`hspace`
: Horizontal space to the left and right of this image, applet, or object. See the hspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`name`
: Form control or object name when submitted with a form. See the name attribute definition in HTML 4.0.

`standby`
: Message to render while loading the object. See the standby attribute definition in HTML 4.0.

`tabIndex`
: Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`type`
: Content type for data downloaded via `data` attribute. See the type attribute definition in HTML 4.0.

`useMap`
: Use client-side image map. See the usemap attribute definition in HTML 4.0.

`vspace`
: Vertical space above and below this image, applet, or object. See the vspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`width`
: Override width. See the width attribute definition in HTML 4.0.

Interface *HTMLParamElement*

Parameters fed to the `OBJECT` element. See the PARAM element definition in HTML 4.0.

IDL Definition

```
interface HTMLParamElement : HTMLElement {
           attribute  DOMString            name;
           attribute  DOMString            type;
           attribute  DOMString            value;
           attribute  DOMString            valueType;
};
```

Attributes

`name`
: The name of a run-time parameter. See the name attribute definition in HTML 4.0.

`type`
: Content type for the `value` attribute when `valuetype` has the value "ref". See the type attribute definition in HTML 4.0.

`value`
: The value of a run-time parameter. See the value attribute definition in HTML 4.0.

`valueType`
: Information about the meaning of the `value` attribute value. See the valuetype attribute definition in HTML 4.0.

Interface *HTMLAppletElement*

An embedded Java applet. See the APPLET element definition in HTML 4.0. This element is deprecated in HTML 4.0.

IDL Definition

```
interface HTMLAppletElement : HTMLElement {
           attribute  DOMString            align;
           attribute  DOMString            alt;
           attribute  DOMString            archive;
           attribute  DOMString            code;
           attribute  DOMString            codeBase;
           attribute  DOMString            height;
           attribute  DOMString            hspace;
           attribute  DOMString            name;
           attribute  DOMString            object;
           attribute  DOMString            vspace;
           attribute  DOMString            width;
};
```

Attributes

`align`
: Aligns this object (vertically or horizontally) with respect to its surrounding text. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`alt`
: Alternate text for user agents not rendering the normal content of this element. See the alt attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`archive`
: Comma-separated archive list. See the archive attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`code`
: Applet class file. See the code attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`codeBase`
: Optional base URI for applet. See the codebase attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`height`
: Override height. See the height attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`hspace`
: Horizontal space to the left and right of this image, applet, or object. See the hspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`name`
: The name of the applet. See the name attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`object`
: Serialized applet file. See the object attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`vspace`
: Vertical space above and below this image, applet, or object. See the vspace attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`width`
: Override width. See the width attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLMapElement*

Client-side image map. See the MAP element definition in HTML 4.0.

IDL Definition

```
interface HTMLMapElement : HTMLElement {
  readonly attribute  HTMLCollection       areas;
           attribute  DOMString            name;
};
```

Attributes

`areas`
: The list of areas defined for the image map.

`name`
: Names the map (for use with `usemap`). See the name attribute definition in HTML 4.0.

Interface *HTMLAreaElement*

Client-side image map area definition. See the AREA element definition in HTML 4.0.

IDL Definition

```
interface HTMLAreaElement : HTMLElement {
           attribute  DOMString            accessKey;
           attribute  DOMString            alt;
           attribute  DOMString            coords;
           attribute  DOMString            href;
           attribute  boolean              noHref;
           attribute  DOMString            shape;
           attribute  long                 tabIndex;
           attribute  DOMString            target;
};
```

Attributes

`accessKey`
: A single character access key to give access to the form control. See the accesskey attribute definition in HTML 4.0.

`alt`
: Alternate text for user agents not rendering the normal content of this element. See the alt attribute definition in HTML 4.0.

`coords`
: Comma-separated list of lengths, defining an active region geometry. See also `shape` for the shape of the region. See the coords attribute definition in HTML 4.0.

`href`
: The URI of the linked resource. See the href attribute definition in HTML 4.0.

`noHref`
: Specifies that this area is inactive, i.e., has no associated action. See the nohref attribute definition in HTML 4.0.

`shape`
: The shape of the active area. The coordinates are given by `coords`. See the shape attribute definition in HTML 4.0.

`tabIndex`
: Index that represents the element's position in the tabbing order. See the tabindex attribute definition in HTML 4.0.

`target`
: Frame to render the resource in. See the target attribute definition in HTML 4.0.

Interface *HTMLScriptElement*

Script statements. See the SCRIPT element definition in HTML 4.0.

IDL Definition

```
interface HTMLScriptElement : HTMLElement {
           attribute  DOMString              text;
           attribute  DOMString              htmlFor;
           attribute  DOMString              event;
           attribute  DOMString              charset;
           attribute  boolean                defer;
           attribute  DOMString              src;
           attribute  DOMString              type;
};
```

Attributes

`text`
: The script content of the element.

`htmlFor`
: *Reserved for future use.*

`event`
: *Reserved for future use.*

`charset`
: The character encoding of the linked resource. See the charset attribute definition in HTML 4.0.

`defer`
: Indicates that the user agent can defer processing of the script. See the defer attribute definition in HTML 4.0.

`src`
: URI designating an external script. See the src attribute definition in HTML 4.0.

`type`
: The content type of the script language. See the type attribute definition in HTML 4.0.

Interface *HTMLTableElement*

The create* and delete* methods on the table allow authors to construct and modify tables. HTML 4.0 specifies that only one of each of the `CAPTION`, `THEAD`, and `TFOOT` elements may exist in a table. Therefore, if one exists, and the createTHead() or createTFoot() method is called, the method returns the existing THead or TFoot element. See the TABLE element definition in HTML 4.0.

IDL Definition

```
interface HTMLTableElement : HTMLElement {
           attribute  HTMLTableCaptionElement caption;
           attribute  HTMLTableSectionElement tHead;
           attribute  HTMLTableSectionElement tFoot;
  readonly attribute  HTMLCollection       rows;
  readonly attribute  HTMLCollection       tBodies;
           attribute  DOMString            align;
           attribute  DOMString            bgColor;
           attribute  DOMString            border;
           attribute  DOMString            cellPadding;
           attribute  DOMString            cellSpacing;
           attribute  DOMString            frame;
           attribute  DOMString            rules;
           attribute  DOMString            summary;
           attribute  DOMString            width;
  HTMLElement               createTHead();
  void                      deleteTHead();
  HTMLElement               createTFoot();
  void                      deleteTFoot();
  HTMLElement               createCaption();
  void                      deleteCaption();
  HTMLElement               insertRow(in long index);
  void                      deleteRow(in long index);
};
```

Attributes

`caption`
: Returns the table's `CAPTION`, or void if none exists.

`tHead`
: Returns the table's `THEAD`, or `null` if none exists.

`tFoot`
: Returns the table's `TFOOT`, or `null` if none exists.

`rows`
: Returns a collection of all the rows in the table, including all in `THEAD`, `TFOOT`, all `TBODY` elements.

`tBodies`
Returns a collection of the defined table bodies.

`align`
Specifies the table's position with respect to the rest of the document. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`bgColor`
Cell background color. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`border`
The width of the border around the table. See the border attribute definition in HTML 4.0.

`cellPadding`
Specifies the horizontal and vertical space between cell content and cell borders. See the cellpadding attribute definition in HTML 4.0.

`cellSpacing`
Specifies the horizontal and vertical separation between cells. See the cellspacing attribute definition in HTML 4.0.

`frame`
Specifies which external table borders to render. See the frame attribute definition in HTML 4.0.

`rules`
Specifies which internal table borders to render. See the rules attribute definition in HTML 4.0.

`summary`
Supplementary description about the purpose or structure of a table. See the summary attribute definition in HTML 4.0.

`width`
Specifies the desired table width. See the width attribute definition in HTML 4.0.

Methods

`createTHead`
Create a table header row or return an existing one.

Return Value

A new table header element (`THEAD`).

This method has no parameters.
This method raises no exceptions.

`deleteTHead`
Delete the header from the table, if one exists.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

`createTFoot`
Create a table footer row or return an existing one.

Return Value

A footer element (`TFOOT`).

This method has no parameters.
This method raises no exceptions.

`deleteTFoot`

Delete the footer from the table, if one exists.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

`createCaption`

Create a new table caption object or return an existing one.

Return Value

A `CAPTION` element.

This method has no parameters.
This method raises no exceptions.

`deleteCaption`

Delete the table caption, if one exists.
This method has no parameters.
This method returns nothing.
This method raises no exceptions.

`insertRow`

Insert a new empty row in the table. *Note.* A table row cannot be empty according to HTML 4.0 Recommendation.

Parameters

`index`	The row number where to insert a new row.

Return Value

The newly created row.

This method raises no exceptions.

`deleteRow`

Delete a table row.

Parameters

`index`	The index of the row to be deleted.

This method returns nothing.
This method raises no exceptions.

Interface *HTMLTableCaptionElement*

Table caption See the CAPTION element definition in HTML 4.0.

IDL Definition

```
interface HTMLTableCaptionElement : HTMLElement {
           attribute  DOMString           align;
};
```

Attributes

`align`
: Caption alignment with respect to the table. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLTableColElement*

Regroups the `COL` and `COLGROUP` elements. See the COL element definition in HTML 4.0.

IDL Definition

```
interface HTMLTableColElement : HTMLElement {
           attribute  DOMString            align;
           attribute  DOMString            ch;
           attribute  DOMString            chOff;
           attribute  long                 span;
           attribute  DOMString            vAlign;
           attribute  DOMString            width;
};
```

Attributes

`align`
: Horizontal alignment of cell data in column. See the align attribute definition in HTML 4.0.

`ch`
: Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff`
: Offset of alignment character. See the charoff attribute definition in HTML 4.0.

`span`
: Indicates the number of columns in a group or affected by a grouping. See the span attribute definition in HTML 4.0.

`vAlign`
: Vertical alignment of cell data in column. See the valign attribute definition in HTML 4.0.

`width`
: Default column width. See the width attribute definition in HTML 4.0.

Interface *HTMLTableSectionElement*

The `THEAD`, `TFOOT`, and `TBODY` elements.

IDL Definition

```
interface HTMLTableSectionElement : HTMLElement {
           attribute  DOMString            align;
           attribute  DOMString            ch;
           attribute  DOMString            chOff;
           attribute  DOMString            vAlign;
  readonly attribute  HTMLCollection       rows;
  HTMLElement                  insertRow(in long index);
  void                         deleteRow(in long index);
};
```

Attributes

`align`

Horizontal alignment of data in cells. See the `align` attribute for HTMLTheadElement for details.

`ch`

Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff`

Offset of alignment character. See the charoff attribute definition in HTML 4.0.

`vAlign`

Vertical alignment of data in cells. See the `valign` attribute for HTMLTheadElement for details.

`rows`

The collection of rows in this table section.

Methods

`insertRow`

Insert a row into this section.

Parameters

`index`	The row number where to insert a new row.

Return Value

The newly created row.

This method raises no exceptions.

`deleteRow`

Delete a row from this section.

Parameters

`index`	The index of the row to be deleted.

This method returns nothing.

This method raises no exceptions.

Interface *HTMLTableRowElement*

A row in a table. See the TR element definition in HTML 4.0.

IDL Definition

```
interface HTMLTableRowElement : HTMLElement {
           attribute  long                  rowIndex;
           attribute  long                  sectionRowIndex;
           attribute  HTMLCollection        cells;
           attribute  DOMString             align;
           attribute  DOMString             bgColor;
           attribute  DOMString             ch;
           attribute  DOMString             chOff;
```

```
             attribute  DOMString            vAlign;
  HTMLElement                 insertCell(in long index);
  void                        deleteCell(in long index);
};
```

Attributes

`rowIndex`
: The index of this row, relative to the entire table.

`sectionRowIndex`
: The index of this row, relative to the current section (`THEAD`, `TFOOT`, or `TBODY`).

`cells`
: The collection of cells in this row.

`align`
: Horizontal alignment of data within cells of this row. See the align attribute definition in HTML 4.0.

`bgColor`
: Background color for rows. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`ch`
: Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff`
: Offset of alignment character. See the charoff attribute definition in HTML 4.0.

`vAlign`
: Vertical alignment of data within cells of this row. See the valign attribute definition in HTML 4.0.

Methods

`insertCell`
: Insert an empty `TD` cell into this row.

Parameters

`index`	The place to insert the cell.

Return Value

The newly created cell.

This method raises no exceptions.

`deleteCell`
: Delete a cell from the current row.

Parameters

`index`	The index of the cell to delete.

This method returns nothing.

This method raises no exceptions.

Interface *HTMLTableCellElement*

The object used to represent the `TH` and `TD` elements. See the TD element definition in HTML 4.0.

IDL Definition

```
interface HTMLTableCellElement : HTMLElement {
           attribute  long                   cellIndex;
           attribute  DOMString              abbr;
           attribute  DOMString              align;
           attribute  DOMString              axis;
           attribute  DOMString              bgColor;
           attribute  DOMString              ch;
           attribute  DOMString              chOff;
           attribute  long                   colSpan;
           attribute  DOMString              headers;
           attribute  DOMString              height;
           attribute  boolean                noWrap;
           attribute  long                   rowSpan;
           attribute  DOMString              scope;
           attribute  DOMString              vAlign;
           attribute  DOMString              width;
};
```

Attributes

`cellIndex`
: The index of this cell in the row.

`abbr`
: Abbreviation for header cells. See the abbr attribute definition in HTML 4.0.

`align`
: Horizontal alignment of data in cell. See the align attribute definition in HTML 4.0.

`axis`
: Names group of related headers. See the axis attribute definition in HTML 4.0.

`bgColor`
: Cell background color. See the bgcolor attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`ch`
: Alignment character for cells in a column. See the char attribute definition in HTML 4.0.

`chOff`
: Offset of alignment character. See the charoff attribute definition in HTML 4.0.

`colSpan`
: Number of columns spanned by cell. See the colspan attribute definition in HTML 4.0.

`headers`
: List of `id` attribute values for header cells. See the headers attribute definition in HTML 4.0.

`height`
: Cell height. See the height attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`noWrap`
: Suppress word wrapping. See the nowrap attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`rowSpan`
: Number of rows spanned by cell. See the rowspan attribute definition in HTML 4.0.

`scope`
: Scope covered by header cells. See the scope attribute definition in HTML 4.0.

`vAlign`
: Vertical alignment of data in cell. See the valign attribute definition in HTML 4.0.

`width`
: Cell width. See the width attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

Interface *HTMLFrameSetElement*

Create a grid of frames. See the FRAMESET element definition in HTML 4.0.

IDL Definition

```
interface HTMLFrameSetElement : HTMLElement {
           attribute  DOMString            cols;
           attribute  DOMString            rows;
};
```

Attributes

`cols`
: The number of columns of frames in the frameset. See the cols attribute definition in HTML 4.0.

`rows`
: The number of rows of frames in the frameset. See the rows attribute definition in HTML 4.0.

Interface *HTMLFrameElement*

Create a frame. See the FRAME element definition in HTML 4.0.

IDL Definition

```
interface HTMLFrameElement : HTMLElement {
           attribute  DOMString            frameBorder;
           attribute  DOMString            longDesc;
           attribute  DOMString            marginHeight;
           attribute  DOMString            marginWidth;
           attribute  DOMString            name;
           attribute  boolean              noResize;
           attribute  DOMString            scrolling;
           attribute  DOMString            src;
};
```

Attributes

`frameBorder`
: Request frame borders. See the frameborder attribute definition in HTML 4.0.

`longDesc`
: URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

`marginHeight`
: Frame margin height, in pixels. See the marginheight attribute definition in HTML 4.0.

`marginWidth`
: Frame margin width, in pixels. See the marginwidth attribute definition in HTML 4.0.

`name`
: The frame name (object of the `target` attribute). See the name attribute definition in HTML 4.0.

`noResize`
: When true, forbid user from resizing frame. See the noresize attribute definition in HTML 4.0.

`scrolling`
: Specify whether or not the frame should have scrollbars. See the scrolling attribute definition in HTML 4.0.

`src`
: A URI designating the initial frame contents. See the src attribute definition in HTML 4.0.

Interface *HTMLIFrameElement*

Inline subwindows. See the IFRAME element definition in HTML 4.0.

IDL Definition

```
interface HTMLIFrameElement : HTMLElement {
           attribute  DOMString            align;
           attribute  DOMString            frameBorder;
           attribute  DOMString            height;
           attribute  DOMString            longDesc;
           attribute  DOMString            marginHeight;
           attribute  DOMString            marginWidth;
           attribute  DOMString            name;
           attribute  DOMString            scrolling;
           attribute  DOMString            src;
           attribute  DOMString            width;
};
```

Attributes

`align`
: Aligns this object (vertically or horizontally) with respect to its surrounding text. See the align attribute definition in HTML 4.0. This attribute is deprecated in HTML 4.0.

`frameBorder`
: Request frame borders. See the frameborder attribute definition in HTML 4.0.

`height`
: Frame height. See the height attribute definition in HTML 4.0.

`longDesc`
: URI designating a long description of this image or frame. See the longdesc attribute definition in HTML 4.0.

`marginHeight`
: Frame margin height, in pixels. See the marginheight attribute definition in HTML 4.0.

`marginWidth`
: Frame margin width, in pixels. See the marginwidth attribute definition in HTML 4.0.

`name`
: The frame name (object of the `target` attribute). See the name attribute definition in HTML 4.0.

`scrolling`
: Specify whether or not the frame should have scrollbars. See the scrolling attribute definition in HTML 4.0.

`src`
: A URI designating the initial frame contents. See the src attribute definition in HTML 4.0.

`width`
: Frame width. See the width attribute definition in HTML 4.0.

2.5.5. Object definitions

Appendix A: Contributors

Members of the DOM Working Group and Interest Group contributing to this specification were:

Lauren Wood, SoftQuad, Inc., *chair*
Arnaud Le Hors, W3C, *W3C staff contact*
Andrew Watson, Object Management Group
Bill Smith, Sun
Chris Lovett, Microsoft
Chris Wilson, Microsoft
David Brownell, Sun
David Singer, IBM
Don Park, invited
Eric Vasilik, Microsoft
Gavin Nicol, INSO
Ian Jacobs, W3C
James Clark, invited
Jared Sorensen, Novell
Jonathan Robie, Texcel
Mike Champion, ArborText
Paul Grosso, ArborText
Peter Sharpe, SoftQuad, Inc.
Phil Karlton, Netscape
Ray Whitmer, iMall
Rich Rollman, Microsoft
Rick Gessner, Netscape
Robert Sutor, IBM
Scott Isaacs, Microsoft
Sharon Adler, INSO
Steve Byrne, JavaSoft
Tim Bray, invited
Tom Pixley, Netscape
Vidur Apparao, Netscape

Appendix A: Contributors

Appendix B: Glossary

Editors
Robert S. Sutor, IBM Research

Several of the following term definitions have been borrowed or modified from similar definitions in other W3C or standards documents. See the links within the definitions for more information.

ancestor
: An *ancestor* node of any node A is any node above A in a tree model of a document, where "above" means "toward the root."

API
: An *API* is an application programming interface, a set of functions or methods used to access some functionality.

child
: A *child* is an immediate descendant node of a node.

client application
: A [client] application is any software that uses the Document Object Model programming interfaces provided by the hosting implementation to accomplish useful work. Some examples of client applications are scripts within an HTML or XML document.

COM
: *COM* is Microsoft's Component Object Model, a technology for building applications from binary software components.

content model
: The *content model* is a simple grammar governing the allowed types of the child elements and the order in which they appear. See [XML]

context
: A *context* specifies an access pattern (or path): a set of interfaces which give you a way to interact with a model. For example, imagine a model with different colored arcs connecting data nodes. A context might be a sheet of colored acetate that is placed over the model allowing you a partial view of the total information in the model.

convenience
: A *convenience method* is an operation on an object that could be accomplished by a program consisting of more basic operations on the object. Convenience methods are usually provided to make the API easier and simpler to use or to allow specific programs to create more optimized implementations for common operations. A similar definition holds for a *convenience property*.

cooked model
: A model for a document that represents the document after it has been manipulated in some way. For example, any combination of any of the following transformations would create a cooked model:

1. Expansion of internal text entities.
2. Expansion of external entities.
3. Model augmentation with style-specified generated text.
4. Execution of style-specified reordering.
5. Execution of scripts.

A browser might only be able to provide access to a cooked model, while an editor might provide access to a cooked or the initial structure model (also known as the *uncooked model*) for a document.

CORBA
: *CORBA* is the *Common Object Request Broker Architecture* from the OMG . This architecture is a collection of objects and libraries that allow the creation of applications containing objects that make and receive requests and responses in a distributed environment.

cursor
A *cursor* is an object representation of a node. It may possess information about context and the path traversed to reach the node.

data model
A *data model* is a collection of descriptions of data structures and their contained fields, together with the operations or functions that manipulate them.

deprecation
When new releases of specifications are released, some older features may be marked as being *deprecated.* This means that new work should not use the features and that although they are supported in the current release, they may not be supported or available in future releases.

descendant
A *descendant* node of any node A is any node below A in a tree model of a document, where "above" means "toward the root."

ECMAScript
The programming language defined by the ECMA-262 standard. As stated in the standard, the originating technology for ECMAScript was JavaScript. Note that in the ECMAScript binding, the word "property" is used in the same sense as the IDL term "attribute."

element
Each document contains one or more elements, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements by an empty-element tag. Each element has a type, identified by name, and may have a set of attributes. Each attribute has a name and a value. [XML]

event propagation, also known as event bubbling
This is the idea that an event can affect one object and a set of related objects. Any of the potentially affected objects can block the event or substitute a different one (upward event propagation). The event is broadcast from the node at which it originates to every parent node.

equivalence
Two nodes are *equivalent* if they have the same node type and same node name. Also, if the nodes contain data, that must be the same. Finally, if the nodes have attributes then collection of attribute names must be the same and the attributes corresponding by name must be equivalent as nodes. Two nodes are *deeply equivalent* if they are *equivalent*, the child node lists are equivalent are equivalent as NodeList objects, and the pairs of equivalent attributes must in fact be deeply equivalent. Two NodeList objects are *equivalent* if they have the same length, and the nodes corresponding by index are deeply equivalent. Two NamedNodeMap objects are *equivalent* if they are have the same length, they have same collection of names, and the nodes corresponding by name in the maps are deeply equivalent. Two DocumentType nodes are *equivalent* if they are equivalent as nodes, have the same names, and have equivalent entities and attributes NamedNodeMap objects.

hosting implementation
A [hosting] implementation is a software module that provides an implementation of the DOM interfaces so that a client application can use them. Some examples of hosting implementations are browsers, editors and document repositories.

HTML
The HyperText Markup Language (*HTML*) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of applications. [HTML 3.2] [HTML4.0]

IDL

An Interface Definition Language (*IDL*) is used to define the interfaces for accessing and operating upon objects. Examples of IDLs are the Object Management Group's IDL , Microsoft's IDL , and Sun's Java IDL .

implementor

Companies, organizations, and individuals that claim to support the Document Object Model as an API for their products.

inheritance

In object-oriented programming, the ability to create new classes (or interfaces) that contain all the methods and properties of another class (or interface), plus additional methods and properties. If class (or interface) D inherits from class (or interface) B, then D is said to be *derived* from B. B is said to be a *base* class (or interface) for D. Some programming languages allow for multiple inheritance, that is, inheritance from more than one class or interface.

initial structure model

Also known as the *raw structure model* or the *uncooked model*, this represents the document before it has been modified by entity expansions, generated text, style-specified reordering, or the execution of scripts. In some implementations, this might correspond to the "initial parse tree" for the document, if it ever exists. Note that a given implementation might not be able to provide access to the initial structure model for a document, though an editor probably would.

interface

An *interface* is a declaration of a set of methods with no information given about their implementation. In object systems that support interfaces and inheritance, interfaces can usually inherit from one another.

language binding

A programming *language binding* for an IDL specification is an implementation of the interfaces in the specification for the given language. For example, a Java language binding for the Document Object Model IDL specification would implement the concrete Java classes that provide the functionality exposed by the interfaces.

method

A *method* is an operation or function that is associated with an object and is allowed to manipulate the object's data.

model

A *model* is the actual data representation for the information at hand. Examples are the structural model and the style model representing the parse structure and the style information associated with a document. The model might be a tree, or a directed graph, or something else.

object model

An *object model* is a collection of descriptions of classes or interfaces, together with their member data, member functions, and class-static operations.

parent

A *parent* is an immediate ancestor node of a node.

root node

The *root node* is the unique node that is not a child of any other node. All other nodes are children or other descendents of the root node. [XML]

sibling

Two nodes are *siblings* if they have the same parent node.

string comparison
: When string matching is required, it is to occur as though the comparison was between 2 sequences of code points from the Unicode 2.0 standard.

tag valid document
: A document is *tag valid* if all begin and end tags are properly balanced and nested.

type valid document
: A document is *type valid* if it conforms to an explicit DTD.

uncooked model
: See initial structure model.

well-formed document
: A document is *well-formed* if it is tag valid and entities are limited to single elements (i.e., single sub-trees).

XML
: Extensible Markup Language (*XML*) is an extremely simple dialect of SGML which is completely described in this document. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML. [XML]

Appendix C: IDL Definitions

This appendix contains the complete OMG IDL for the Level 1 Document Object Model definitions. The definitions are divided into Core and HTML.

The IDL files are also available as: http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/idl.zip

C.1: Document Object Model Level 1 Core

This section contains the OMG IDL definitions for the interfaces in the Core Document Object Model specification, including the extended (XML) interfaces.

```
exception DOMException {
  unsigned short   code;
};

// ExceptionCode
const unsigned short      INDEX_SIZE_ERR     = 1;
const unsigned short      DOMSTRING_SIZE_ERR = 2;
const unsigned short      HIERARCHY_REQUEST_ERR = 3;
const unsigned short      WRONG_DOCUMENT_ERR = 4;
const unsigned short      INVALID_CHARACTER_ERR = 5;
const unsigned short      NO_DATA_ALLOWED_ERR = 6;
const unsigned short      NO_MODIFICATION_ALLOWED_ERR = 7;
const unsigned short      NOT_FOUND_ERR      = 8;
const unsigned short      NOT_SUPPORTED_ERR  = 9;
const unsigned short      INUSE_ATTRIBUTE_ERR = 10;
```


```
// ExceptionCode
const unsigned short      INDEX_SIZE_ERR     = 1;
const unsigned short      DOMSTRING_SIZE_ERR = 2;
const unsigned short      HIERARCHY_REQUEST_ERR = 3;
const unsigned short      WRONG_DOCUMENT_ERR = 4;
const unsigned short      INVALID_CHARACTER_ERR = 5;
const unsigned short      NO_DATA_ALLOWED_ERR = 6;
const unsigned short      NO_MODIFICATION_ALLOWED_ERR = 7;
const unsigned short      NOT_FOUND_ERR      = 8;
const unsigned short      NOT_SUPPORTED_ERR  = 9;
const unsigned short      INUSE_ATTRIBUTE_ERR = 10;
```


```
interface DOMImplementation {
  boolean                   hasFeature(in DOMString feature,
                                       in DOMString version);
};

interface DocumentFragment : Node {
};

interface Document : Node {
  readonly attribute  DocumentType         doctype;
  readonly attribute  DOMImplementation    implementation;
  readonly attribute  Element              documentElement;
  Element                   createElement(in DOMString tagName)
```

```
                                        raises(DOMException);
  DocumentFragment           createDocumentFragment();
  Text                       createTextNode(in DOMString data);
  Comment                    createComment(in DOMString data);
  CDATASection               createCDATASection(in DOMString data)
                                               raises(DOMException);
  ProcessingInstruction      createProcessingInstruction(in DOMString target,
                                                         in DOMString data)
                                                         raises(DOMException);
  Attr                       createAttribute(in DOMString name)
                                            raises(DOMException);
  EntityReference            createEntityReference(in DOMString name)
                                                  raises(DOMException);
  NodeList                   getElementsByTagName(in DOMString tagname);
};

interface Node {
  // NodeType
  const unsigned short       ELEMENT_NODE       = 1;
  const unsigned short       ATTRIBUTE_NODE     = 2;
  const unsigned short       TEXT_NODE          = 3;
  const unsigned short       CDATA_SECTION_NODE = 4;
  const unsigned short       ENTITY_REFERENCE_NODE = 5;
  const unsigned short       ENTITY_NODE        = 6;
  const unsigned short       PROCESSING_INSTRUCTION_NODE = 7;
  const unsigned short       COMMENT_NODE       = 8;
  const unsigned short       DOCUMENT_NODE      = 9;
  const unsigned short       DOCUMENT_TYPE_NODE = 10;
  const unsigned short       DOCUMENT_FRAGMENT_NODE = 11;
  const unsigned short       NOTATION_NODE      = 12;

  readonly attribute  DOMString             nodeName;
           attribute  DOMString             nodeValue;
                                                  // raises(DOMException) on setting
                                                  // raises(DOMException) on retrieval
  readonly attribute  unsigned short        nodeType;
  readonly attribute  Node                  parentNode;
  readonly attribute  NodeList              childNodes;
  readonly attribute  Node                  firstChild;
  readonly attribute  Node                  lastChild;
  readonly attribute  Node                  previousSibling;
  readonly attribute  Node                  nextSibling;
  readonly attribute  NamedNodeMap          attributes;
  readonly attribute  Document              ownerDocument;
  Node                       insertBefore(in Node newChild,
                                          in Node refChild)
                                          raises(DOMException);
  Node                       replaceChild(in Node newChild,
                                          in Node oldChild)
                                          raises(DOMException);
  Node                       removeChild(in Node oldChild)
                                         raises(DOMException);
  Node                       appendChild(in Node newChild)
                                         raises(DOMException);
  boolean                    hasChildNodes();
  Node                       cloneNode(in boolean deep);
};
```

```
interface NodeList {
  Node                      item(in unsigned long index);
  readonly attribute  unsigned long        length;
};

interface NamedNodeMap {
  Node                      getNamedItem(in DOMString name);
  Node                      setNamedItem(in Node arg)
                                         raises(DOMException);
  Node                      removeNamedItem(in DOMString name)
                                            raises(DOMException);
  Node                      item(in unsigned long index);
  readonly attribute  unsigned long        length;
};

interface CharacterData : Node {
           attribute  DOMString            data;
                                // raises(DOMException) on setting
                                // raises(DOMException) on retrieval
  readonly attribute  unsigned long        length;
  DOMString                 substringData(in unsigned long offset,
                                          in unsigned long count)
                                          raises(DOMException);
  void                      appendData(in DOMString arg)
                                       raises(DOMException);
  void                      insertData(in unsigned long offset,
                                       in DOMString arg)
                                       raises(DOMException);
  void                      deleteData(in unsigned long offset,
                                       in unsigned long count)
                                       raises(DOMException);
  void                      replaceData(in unsigned long offset,
                                        in unsigned long count,
                                        in DOMString arg)
                                        raises(DOMException);
};

interface Attr : Node {
  readonly attribute  DOMString            name;
  readonly attribute  boolean              specified;
           attribute  DOMString            value;
};

interface Element : Node {
  readonly attribute  DOMString            tagName;
  DOMString                 getAttribute(in DOMString name);
  void                      setAttribute(in DOMString name,
                                         in DOMString value)
                                         raises(DOMException);
  void                      removeAttribute(in DOMString name)
                                            raises(DOMException);
  Attr                      getAttributeNode(in DOMString name);
  Attr                      setAttributeNode(in Attr newAttr)
                                             raises(DOMException);
  Attr                      removeAttributeNode(in Attr oldAttr)
                                                raises(DOMException);
```

```
  NodeList                  getElementsByTagName(in DOMString name);
  void                      normalize();
};

interface Text : CharacterData {
  Text                      splitText(in unsigned long offset)
                                      raises(DOMException);
};

interface Comment : CharacterData {
};

interface CDATASection : Text {
};

interface DocumentType : Node {
  readonly attribute  DOMString            name;
  readonly attribute  NamedNodeMap         entities;
  readonly attribute  NamedNodeMap         notations;
};

interface Notation : Node {
  readonly attribute  DOMString            publicId;
  readonly attribute  DOMString            systemId;
};

interface Entity : Node {
  readonly attribute  DOMString            publicId;
  readonly attribute  DOMString            systemId;
  readonly attribute  DOMString            notationName;
};

interface EntityReference : Node {
};

interface ProcessingInstruction : Node {
  readonly attribute  DOMString            target;
           attribute  DOMString            data;
                                  // raises(DOMException) on setting
};
```

C.2: Document Object Model Level 1 HTML

```
interface HTMLCollection {
  readonly attribute  unsigned long        length;
  Node                      item(in unsigned long index);
  Node                      namedItem(in DOMString name);
};

interface HTMLDocument : Document {
           attribute  DOMString            title;
  readonly attribute  DOMString            referrer;
  readonly attribute  DOMString            domain;
  readonly attribute  DOMString            URL;
           attribute  HTMLElement          body;
  readonly attribute  HTMLCollection       images;
```

```
    readonly attribute  HTMLCollection       applets;
    readonly attribute  HTMLCollection       links;
    readonly attribute  HTMLCollection       forms;
    readonly attribute  HTMLCollection       anchors;
             attribute  DOMString            cookie;
    void                      open();
    void                      close();
    void                      write(in DOMString text);
    void                      writeln(in DOMString text);
    Element                   getElementById(in DOMString elementId);
    NodeList                  getElementsByName(in DOMString elementName);
  };

  interface HTMLElement : Element {
             attribute  DOMString            id;
             attribute  DOMString            title;
             attribute  DOMString            lang;
             attribute  DOMString            dir;
             attribute  DOMString            className;
  };

  interface HTMLHtmlElement : HTMLElement {
             attribute  DOMString            version;
  };

  interface HTMLHeadElement : HTMLElement {
             attribute  DOMString            profile;
  };

  interface HTMLLinkElement : HTMLElement {
             attribute  boolean              disabled;
             attribute  DOMString            charset;
             attribute  DOMString            href;
             attribute  DOMString            hreflang;
             attribute  DOMString            media;
             attribute  DOMString            rel;
             attribute  DOMString            rev;
             attribute  DOMString            target;
             attribute  DOMString            type;
  };

  interface HTMLTitleElement : HTMLElement {
             attribute  DOMString            text;
  };

  interface HTMLMetaElement : HTMLElement {
             attribute  DOMString            content;
             attribute  DOMString            httpEquiv;
             attribute  DOMString            name;
             attribute  DOMString            scheme;
  };

  interface HTMLBaseElement : HTMLElement {
             attribute  DOMString            href;
             attribute  DOMString            target;
  };
```

```
interface HTMLIsIndexElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  DOMString            prompt;
};

interface HTMLStyleElement : HTMLElement {
           attribute  boolean              disabled;
           attribute  DOMString            media;
           attribute  DOMString            type;
};

interface HTMLBodyElement : HTMLElement {
           attribute  DOMString            aLink;
           attribute  DOMString            background;
           attribute  DOMString            bgColor;
           attribute  DOMString            link;
           attribute  DOMString            text;
           attribute  DOMString            vLink;
};

interface HTMLFormElement : HTMLElement {
  readonly attribute  HTMLCollection       elements;
  readonly attribute  long                 length;
           attribute  DOMString            name;
           attribute  DOMString            acceptCharset;
           attribute  DOMString            action;
           attribute  DOMString            enctype;
           attribute  DOMString            method;
           attribute  DOMString            target;
  void                      submit();
  void                      reset();
};

interface HTMLSelectElement : HTMLElement {
  readonly attribute  DOMString            type;
           attribute  long                 selectedIndex;
           attribute  DOMString            value;
  readonly attribute  long                 length;
  readonly attribute  HTMLFormElement      form;
  readonly attribute  HTMLCollection       options;
           attribute  boolean              disabled;
           attribute  boolean              multiple;
           attribute  DOMString            name;
           attribute  long                 size;
           attribute  long                 tabIndex;
  void                      add(in HTMLElement element,
                                in HTMLElement before);
  void                      remove(in long index);
  void                      blur();
  void                      focus();
};

interface HTMLOptGroupElement : HTMLElement {
           attribute  boolean              disabled;
           attribute  DOMString            label;
};
```

```
interface HTMLOptionElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  boolean              defaultSelected;
  readonly attribute  DOMString            text;
           attribute  long                 index;
           attribute  boolean              disabled;
           attribute  DOMString            label;
  readonly attribute  boolean              selected;
           attribute  DOMString            value;
};

interface HTMLInputElement : HTMLElement {
           attribute  DOMString            defaultValue;
           attribute  boolean              defaultChecked;
  readonly attribute  HTMLFormElement      form;
           attribute  DOMString            accept;
           attribute  DOMString            accessKey;
           attribute  DOMString            align;
           attribute  DOMString            alt;
           attribute  boolean              checked;
           attribute  boolean              disabled;
           attribute  long                 maxLength;
           attribute  DOMString            name;
           attribute  boolean              readOnly;
           attribute  DOMString            size;
           attribute  DOMString            src;
           attribute  long                 tabIndex;
  readonly attribute  DOMString            type;
           attribute  DOMString            useMap;
           attribute  DOMString            value;
  void                      blur();
  void                      focus();
  void                      select();
  void                      click();
};

interface HTMLTextAreaElement : HTMLElement {
           attribute  DOMString            defaultValue;
  readonly attribute  HTMLFormElement      form;
           attribute  DOMString            accessKey;
           attribute  long                 cols;
           attribute  boolean              disabled;
           attribute  DOMString            name;
           attribute  boolean              readOnly;
           attribute  long                 rows;
           attribute  long                 tabIndex;
  readonly attribute  DOMString            type;
           attribute  DOMString            value;
  void                      blur();
  void                      focus();
  void                      select();
};

interface HTMLButtonElement : HTMLElement {
  readonly attribute  HTMLFormElement      form;
           attribute  DOMString            accessKey;
           attribute  boolean              disabled;
```

```
           attribute  DOMString              name;
           attribute  long                   tabIndex;
  readonly attribute  DOMString              type;
           attribute  DOMString              value;
};

interface HTMLLabelElement : HTMLElement {
  readonly attribute  HTMLFormElement        form;
           attribute  DOMString              accessKey;
           attribute  DOMString              htmlFor;
};

interface HTMLFieldSetElement : HTMLElement {
  readonly attribute  HTMLFormElement        form;
};

interface HTMLLegendElement : HTMLElement {
  readonly attribute  HTMLFormElement        form;
           attribute  DOMString              accessKey;
           attribute  DOMString              align;
};

interface HTMLUListElement : HTMLElement {
           attribute  boolean                compact;
           attribute  DOMString              type;
};

interface HTMLOListElement : HTMLElement {
           attribute  boolean                compact;
           attribute  long                   start;
           attribute  DOMString              type;
};

interface HTMLDListElement : HTMLElement {
           attribute  boolean                compact;
};

interface HTMLDirectoryElement : HTMLElement {
           attribute  boolean                compact;
};

interface HTMLMenuElement : HTMLElement {
           attribute  boolean                compact;
};

interface HTMLLIElement : HTMLElement {
           attribute  DOMString              type;
           attribute  long                   value;
};

interface HTMLBlockquoteElement : HTMLElement {
           attribute  DOMString              cite;
};

interface HTMLDivElement : HTMLElement {
           attribute  DOMString              align;
};
```

```
interface HTMLParagraphElement : HTMLElement {
          attribute  DOMString            align;
};

interface HTMLHeadingElement : HTMLElement {
          attribute  DOMString            align;
};

interface HTMLQuoteElement : HTMLElement {
          attribute  DOMString            cite;
};

interface HTMLPreElement : HTMLElement {
          attribute  long                 width;
};

interface HTMLBRElement : HTMLElement {
          attribute  DOMString            clear;
};

interface HTMLBaseFontElement : HTMLElement {
          attribute  DOMString            color;
          attribute  DOMString            face;
          attribute  DOMString            size;
};

interface HTMLFontElement : HTMLElement {
          attribute  DOMString            color;
          attribute  DOMString            face;
          attribute  DOMString            size;
};

interface HTMLHRElement : HTMLElement {
          attribute  DOMString            align;
          attribute  boolean              noShade;
          attribute  DOMString            size;
          attribute  DOMString            width;
};

interface HTMLModElement : HTMLElement {
          attribute  DOMString            cite;
          attribute  DOMString            dateTime;
};

interface HTMLAnchorElement : HTMLElement {
          attribute  DOMString            accessKey;
          attribute  DOMString            charset;
          attribute  DOMString            coords;
          attribute  DOMString            href;
          attribute  DOMString            hreflang;
          attribute  DOMString            name;
          attribute  DOMString            rel;
          attribute  DOMString            rev;
          attribute  DOMString            shape;
          attribute  long                 tabIndex;
          attribute  DOMString            target;
```

```
          attribute  DOMString           type;
  void                      blur();
  void                      focus();
};

interface HTMLImageElement : HTMLElement {
          attribute  DOMString           lowSrc;
          attribute  DOMString           name;
          attribute  DOMString           align;
          attribute  DOMString           alt;
          attribute  DOMString           border;
          attribute  DOMString           height;
          attribute  DOMString           hspace;
          attribute  boolean             isMap;
          attribute  DOMString           longDesc;
          attribute  DOMString           src;
          attribute  DOMString           useMap;
          attribute  DOMString           vspace;
          attribute  DOMString           width;
};

interface HTMLObjectElement : HTMLElement {
  readonly attribute  HTMLFormElement     form;
          attribute  DOMString           code;
          attribute  DOMString           align;
          attribute  DOMString           archive;
          attribute  DOMString           border;
          attribute  DOMString           codeBase;
          attribute  DOMString           codeType;
          attribute  DOMString           data;
          attribute  boolean             declare;
          attribute  DOMString           height;
          attribute  DOMString           hspace;
          attribute  DOMString           name;
          attribute  DOMString           standby;
          attribute  long                tabIndex;
          attribute  DOMString           type;
          attribute  DOMString           useMap;
          attribute  DOMString           vspace;
          attribute  DOMString           width;
};

interface HTMLParamElement : HTMLElement {
          attribute  DOMString           name;
          attribute  DOMString           type;
          attribute  DOMString           value;
          attribute  DOMString           valueType;
};

interface HTMLAppletElement : HTMLElement {
          attribute  DOMString           align;
          attribute  DOMString           alt;
          attribute  DOMString           archive;
          attribute  DOMString           code;
          attribute  DOMString           codeBase;
          attribute  DOMString           height;
          attribute  DOMString           hspace;
```

```
            attribute  DOMString            name;
            attribute  DOMString            object;
            attribute  DOMString            vspace;
            attribute  DOMString            width;
};

interface HTMLMapElement : HTMLElement {
  readonly attribute  HTMLCollection       areas;
           attribute  DOMString            name;
};

interface HTMLAreaElement : HTMLElement {
           attribute  DOMString            accessKey;
           attribute  DOMString            alt;
           attribute  DOMString            coords;
           attribute  DOMString            href;
           attribute  boolean              noHref;
           attribute  DOMString            shape;
           attribute  long                 tabIndex;
           attribute  DOMString            target;
};

interface HTMLScriptElement : HTMLElement {
           attribute  DOMString            text;
           attribute  DOMString            htmlFor;
           attribute  DOMString            event;
           attribute  DOMString            charset;
           attribute  boolean              defer;
           attribute  DOMString            src;
           attribute  DOMString            type;
};

interface HTMLTableElement : HTMLElement {
           attribute  HTMLTableCaptionElement caption;
           attribute  HTMLTableSectionElement tHead;
           attribute  HTMLTableSectionElement tFoot;
  readonly attribute  HTMLCollection       rows;
  readonly attribute  HTMLCollection       tBodies;
           attribute  DOMString            align;
           attribute  DOMString            bgColor;
           attribute  DOMString            border;
           attribute  DOMString            cellPadding;
           attribute  DOMString            cellSpacing;
           attribute  DOMString            frame;
           attribute  DOMString            rules;
           attribute  DOMString            summary;
           attribute  DOMString            width;
  HTMLElement                createTHead();
  void                       deleteTHead();
  HTMLElement                createTFoot();
  void                       deleteTFoot();
  HTMLElement                createCaption();
  void                       deleteCaption();
  HTMLElement                insertRow(in long index);
  void                       deleteRow(in long index);
};
```

```
interface HTMLTableCaptionElement : HTMLElement {
           attribute  DOMString           align;
};

interface HTMLTableColElement : HTMLElement {
           attribute  DOMString           align;
           attribute  DOMString           ch;
           attribute  DOMString           chOff;
           attribute  long                span;
           attribute  DOMString           vAlign;
           attribute  DOMString           width;
};

interface HTMLTableSectionElement : HTMLElement {
           attribute  DOMString           align;
           attribute  DOMString           ch;
           attribute  DOMString           chOff;
           attribute  DOMString           vAlign;
  readonly attribute  HTMLCollection      rows;
  HTMLElement                insertRow(in long index);
  void                       deleteRow(in long index);
};

interface HTMLTableRowElement : HTMLElement {
           attribute  long                rowIndex;
           attribute  long                sectionRowIndex;
           attribute  HTMLCollection      cells;
           attribute  DOMString           align;
           attribute  DOMString           bgColor;
           attribute  DOMString           ch;
           attribute  DOMString           chOff;
           attribute  DOMString           vAlign;
  HTMLElement                insertCell(in long index);
  void                       deleteCell(in long index);
};

interface HTMLTableCellElement : HTMLElement {
           attribute  long                cellIndex;
           attribute  DOMString           abbr;
           attribute  DOMString           align;
           attribute  DOMString           axis;
           attribute  DOMString           bgColor;
           attribute  DOMString           ch;
           attribute  DOMString           chOff;
           attribute  long                colSpan;
           attribute  DOMString           headers;
           attribute  DOMString           height;
           attribute  boolean             noWrap;
           attribute  long                rowSpan;
           attribute  DOMString           scope;
           attribute  DOMString           vAlign;
           attribute  DOMString           width;
};

interface HTMLFrameSetElement : HTMLElement {
           attribute  DOMString           cols;
           attribute  DOMString           rows;
```

```
};

interface HTMLFrameElement : HTMLElement {
           attribute  DOMString            frameBorder;
           attribute  DOMString            longDesc;
           attribute  DOMString            marginHeight;
           attribute  DOMString            marginWidth;
           attribute  DOMString            name;
           attribute  boolean              noResize;
           attribute  DOMString            scrolling;
           attribute  DOMString            src;
};

interface HTMLIFrameElement : HTMLElement {
           attribute  DOMString            align;
           attribute  DOMString            frameBorder;
           attribute  DOMString            height;
           attribute  DOMString            longDesc;
           attribute  DOMString            marginHeight;
           attribute  DOMString            marginWidth;
           attribute  DOMString            name;
           attribute  DOMString            scrolling;
           attribute  DOMString            src;
           attribute  DOMString            width;
};
```

Appendix D: Java Language Binding

This appendix contains the complete Java binding for the Level 1 Document Object Model. The definitions are divided into Core and HTML.

The Java files are also available as
http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/java-binding.zip

D.1: Document Object Model Level 1 Core

```
public abstract class DOMException extends RuntimeException {
  public DOMException(short code, String message) {
     super(message);
     this.code = code;
  }
  public short   code;
  // ExceptionCode
  public static final short           INDEX_SIZE_ERR       = 1;
  public static final short           DOMSTRING_SIZE_ERR   = 2;
  public static final short           HIERARCHY_REQUEST_ERR = 3;
  public static final short           WRONG_DOCUMENT_ERR   = 4;
  public static final short           INVALID_CHARACTER_ERR = 5;
  public static final short           NO_DATA_ALLOWED_ERR  = 6;
  public static final short           NO_MODIFICATION_ALLOWED_ERR = 7;
  public static final short           NOT_FOUND_ERR        = 8;
  public static final short           NOT_SUPPORTED_ERR    = 9;
  public static final short           INUSE_ATTRIBUTE_ERR  = 10;

}

// ExceptionCode
public static final short           INDEX_SIZE_ERR       = 1;
public static final short           DOMSTRING_SIZE_ERR   = 2;
public static final short           HIERARCHY_REQUEST_ERR = 3;
public static final short           WRONG_DOCUMENT_ERR   = 4;
public static final short           INVALID_CHARACTER_ERR = 5;
public static final short           NO_DATA_ALLOWED_ERR  = 6;
public static final short           NO_MODIFICATION_ALLOWED_ERR = 7;
public static final short           NOT_FOUND_ERR        = 8;
public static final short           NOT_SUPPORTED_ERR    = 9;
public static final short           INUSE_ATTRIBUTE_ERR  = 10;

}

public interface DOMImplementation {
  public boolean            hasFeature(String feature,
                                       String version);
}

public interface DocumentFragment extends Node {
}

public interface Document extends Node {
  public DocumentType       getDoctype();
```

```
  public DOMImplementation  getImplementation();
  public Element            getDocumentElement();
  public Element            createElement(String tagName)
                                 throws DOMException;
  public DocumentFragment   createDocumentFragment();
  public Text               createTextNode(String data);
  public Comment            createComment(String data);
  public CDATASection       createCDATASection(String data)
                                      throws DOMException;
  public ProcessingInstruction createProcessingInstruction(String target,
                                                 String data)
                                                 throws DOMException;
  public Attr               createAttribute(String name)
                                 throws DOMException;
  public EntityReference    createEntityReference(String name)
                                      throws DOMException;
  public NodeList           getElementsByTagName(String tagname);
}

public interface Node {
  // NodeType
  public static final short          ELEMENT_NODE          = 1;
  public static final short          ATTRIBUTE_NODE        = 2;
  public static final short          TEXT_NODE             = 3;
  public static final short          CDATA_SECTION_NODE    = 4;
  public static final short          ENTITY_REFERENCE_NODE = 5;
  public static final short          ENTITY_NODE           = 6;
  public static final short          PROCESSING_INSTRUCTION_NODE = 7;
  public static final short          COMMENT_NODE          = 8;
  public static final short          DOCUMENT_NODE         = 9;
  public static final short          DOCUMENT_TYPE_NODE    = 10;
  public static final short          DOCUMENT_FRAGMENT_NODE = 11;
  public static final short          NOTATION_NODE         = 12;

  public String             getNodeName();
  public String             getNodeValue()
                                      throws DOMException;
  public void               setNodeValue(String nodeValue)
                                      throws DOMException;
  public short              getNodeType();
  public Node               getParentNode();
  public NodeList           getChildNodes();
  public Node               getFirstChild();
  public Node               getLastChild();
  public Node               getPreviousSibling();
  public Node               getNextSibling();
  public NamedNodeMap       getAttributes();
  public Document           getOwnerDocument();
  public Node               insertBefore(Node newChild,
                                 Node refChild)
                                 throws DOMException;
  public Node               replaceChild(Node newChild,
                                 Node oldChild)
                                 throws DOMException;
  public Node               removeChild(Node oldChild)
                                 throws DOMException;
  public Node               appendChild(Node newChild)
```

```
                                    throws DOMException;
  public boolean            hasChildNodes();
  public Node               cloneNode(boolean deep);
}

public interface NodeList {
  public Node               item(int index);
  public int                getLength();
}

public interface NamedNodeMap {
  public Node               getNamedItem(String name);
  public Node               setNamedItem(Node arg)
                                         throws DOMException;
  public Node               removeNamedItem(String name)
                                            throws DOMException;
  public Node               item(int index);
  public int                getLength();
}

public interface CharacterData extends Node {
  public String             getData()
                                 throws DOMException;
  public void               setData(String data)
                                 throws DOMException;
  public int                getLength();
  public String             substringData(int offset,
                                          int count)
                                          throws DOMException;
  public void               appendData(String arg)
                                       throws DOMException;
  public void               insertData(int offset,
                                       String arg)
                                       throws DOMException;
  public void               deleteData(int offset,
                                       int count)
                                       throws DOMException;
  public void               replaceData(int offset,
                                        int count,
                                        String arg)
                                        throws DOMException;
}

public interface Attr extends Node {
  public String             getName();
  public boolean            getSpecified();
  public String             getValue();
  public void               setValue(String value);
}

public interface Element extends Node {
  public String             getTagName();
  public String             getAttribute(String name);
  public void               setAttribute(String name,
                                         String value)
                                         throws DOMException;
  public void               removeAttribute(String name)
```

```
                                           throws DOMException;
  public Attr                 getAttributeNode(String name);
  public Attr                 setAttributeNode(Attr newAttr)
                                            throws DOMException;
  public Attr                 removeAttributeNode(Attr oldAttr)
                                               throws DOMException;
  public NodeList             getElementsByTagName(String name);
  public void                 normalize();
}

public interface Text extends CharacterData {
  public Text                 splitText(int offset)
                                        throws DOMException;
}

public interface Comment extends CharacterData {
}

public interface CDATASection extends Text {
}

public interface DocumentType extends Node {
  public String               getName();
  public NamedNodeMap         getEntities();
  public NamedNodeMap         getNotations();
}

public interface Notation extends Node {
  public String               getPublicId();
  public String               getSystemId();
}

public interface Entity extends Node {
  public String               getPublicId();
  public String               getSystemId();
  public String               getNotationName();
}

public interface EntityReference extends Node {
}

public interface ProcessingInstruction extends Node {
  public String               getTarget();
  public String               getData();
  public void                 setData(String data)
                                        throws DOMException;
}
```

D.2: Document Object Model Level 1 HTML

```
public interface HTMLCollection {
  public int                  getLength();
  public Node                 item(int index);
  public Node                 namedItem(String name);
}
```

```
public interface HTMLDocument extends Document {
  public String              getTitle();
  public void                setTitle(String title);
  public String              getReferrer();
  public String              getDomain();
  public String              getURL();
  public HTMLElement         getBody();
  public void                setBody(HTMLElement body);
  public HTMLCollection      getImages();
  public HTMLCollection      getApplets();
  public HTMLCollection      getLinks();
  public HTMLCollection      getForms();
  public HTMLCollection      getAnchors();
  public String              getCookie();
  public void                setCookie(String cookie);
  public void                open();
  public void                close();
  public void                write(String text);
  public void                writeln(String text);
  public Element             getElementById(String elementId);
  public NodeList            getElementsByName(String elementName);
}

public interface HTMLElement extends Element {
  public String              getId();
  public void                setId(String id);
  public String              getTitle();
  public void                setTitle(String title);
  public String              getLang();
  public void                setLang(String lang);
  public String              getDir();
  public void                setDir(String dir);
  public String              getClassName();
  public void                setClassName(String className);
}

public interface HTMLHtmlElement extends HTMLElement {
  public String              getVersion();
  public void                setVersion(String version);
}

public interface HTMLHeadElement extends HTMLElement {
  public String              getProfile();
  public void                setProfile(String profile);
}

public interface HTMLLinkElement extends HTMLElement {
  public boolean             getDisabled();
  public void                setDisabled(boolean disabled);
  public String              getCharset();
  public void                setCharset(String charset);
  public String              getHref();
  public void                setHref(String href);
  public String              getHreflang();
  public void                setHreflang(String hreflang);
  public String              getMedia();
  public void                setMedia(String media);
```

```
  public String              getRel();
  public void                setRel(String rel);
  public String              getRev();
  public void                setRev(String rev);
  public String              getTarget();
  public void                setTarget(String target);
  public String              getType();
  public void                setType(String type);
}

public interface HTMLTitleElement extends HTMLElement {
  public String              getText();
  public void                setText(String text);
}

public interface HTMLMetaElement extends HTMLElement {
  public String              getContent();
  public void                setContent(String content);
  public String              getHttpEquiv();
  public void                setHttpEquiv(String httpEquiv);
  public String              getName();
  public void                setName(String name);
  public String              getScheme();
  public void                setScheme(String scheme);
}

public interface HTMLBaseElement extends HTMLElement {
  public String              getHref();
  public void                setHref(String href);
  public String              getTarget();
  public void                setTarget(String target);
}

public interface HTMLIsIndexElement extends HTMLElement {
  public HTMLFormElement     getForm();
  public String              getPrompt();
  public void                setPrompt(String prompt);
}

public interface HTMLStyleElement extends HTMLElement {
  public boolean             getDisabled();
  public void                setDisabled(boolean disabled);
  public String              getMedia();
  public void                setMedia(String media);
  public String              getType();
  public void                setType(String type);
}

public interface HTMLBodyElement extends HTMLElement {
  public String              getALink();
  public void                setALink(String aLink);
  public String              getBackground();
  public void                setBackground(String background);
  public String              getBgColor();
  public void                setBgColor(String bgColor);
  public String              getLink();
  public void                setLink(String link);
```

```
  public String             getText();
  public void               setText(String text);
  public String             getVLink();
  public void               setVLink(String vLink);
}

public interface HTMLFormElement extends HTMLElement {
  public HTMLCollection     getElements();
  public int                getLength();
  public String             getName();
  public void               setName(String name);
  public String             getAcceptCharset();
  public void               setAcceptCharset(String acceptCharset);
  public String             getAction();
  public void               setAction(String action);
  public String             getEnctype();
  public void               setEnctype(String enctype);
  public String             getMethod();
  public void               setMethod(String method);
  public String             getTarget();
  public void               setTarget(String target);
  public void               submit();
  public void               reset();
}

public interface HTMLSelectElement extends HTMLElement {
  public String             getType();
  public int                getSelectedIndex();
  public void               setSelectedIndex(int selectedIndex);
  public String             getValue();
  public void               setValue(String value);
  public int                getLength();
  public HTMLFormElement    getForm();
  public HTMLCollection     getOptions();
  public boolean            getDisabled();
  public void               setDisabled(boolean disabled);
  public boolean            getMultiple();
  public void               setMultiple(boolean multiple);
  public String             getName();
  public void               setName(String name);
  public int                getSize();
  public void               setSize(int size);
  public int                getTabIndex();
  public void               setTabIndex(int tabIndex);
  public void               add(HTMLElement element,
                                HTMLElement before);
  public void               remove(int index);
  public void               blur();
  public void               focus();
}

public interface HTMLOptGroupElement extends HTMLElement {
  public boolean            getDisabled();
  public void               setDisabled(boolean disabled);
  public String             getLabel();
  public void               setLabel(String label);
}
```

```
public interface HTMLOptionElement extends HTMLElement {
  public HTMLFormElement     getForm();
  public boolean             getDefaultSelected();
  public void                setDefaultSelected(boolean defaultSelected);
  public String              getText();
  public int                 getIndex();
  public void                setIndex(int index);
  public boolean             getDisabled();
  public void                setDisabled(boolean disabled);
  public String              getLabel();
  public void                setLabel(String label);
  public boolean             getSelected();
  public String              getValue();
  public void                setValue(String value);
}

public interface HTMLInputElement extends HTMLElement {
  public String              getDefaultValue();
  public void                setDefaultValue(String defaultValue);
  public boolean             getDefaultChecked();
  public void                setDefaultChecked(boolean defaultChecked);
  public HTMLFormElement     getForm();
  public String              getAccept();
  public void                setAccept(String accept);
  public String              getAccessKey();
  public void                setAccessKey(String accessKey);
  public String              getAlign();
  public void                setAlign(String align);
  public String              getAlt();
  public void                setAlt(String alt);
  public boolean             getChecked();
  public void                setChecked(boolean checked);
  public boolean             getDisabled();
  public void                setDisabled(boolean disabled);
  public int                 getMaxLength();
  public void                setMaxLength(int maxLength);
  public String              getName();
  public void                setName(String name);
  public boolean             getReadOnly();
  public void                setReadOnly(boolean readOnly);
  public String              getSize();
  public void                setSize(String size);
  public String              getSrc();
  public void                setSrc(String src);
  public int                 getTabIndex();
  public void                setTabIndex(int tabIndex);
  public String              getType();
  public String              getUseMap();
  public void                setUseMap(String useMap);
  public String              getValue();
  public void                setValue(String value);
  public void                blur();
  public void                focus();
  public void                select();
  public void                click();
}
```

```
public interface HTMLTextAreaElement extends HTMLElement {
  public String             getDefaultValue();
  public void               setDefaultValue(String defaultValue);
  public HTMLFormElement    getForm();
  public String             getAccessKey();
  public void               setAccessKey(String accessKey);
  public int                getCols();
  public void               setCols(int cols);
  public boolean            getDisabled();
  public void               setDisabled(boolean disabled);
  public String             getName();
  public void               setName(String name);
  public boolean            getReadOnly();
  public void               setReadOnly(boolean readOnly);
  public int                getRows();
  public void               setRows(int rows);
  public int                getTabIndex();
  public void               setTabIndex(int tabIndex);
  public String             getType();
  public String             getValue();
  public void               setValue(String value);
  public void               blur();
  public void               focus();
  public void               select();
}

public interface HTMLButtonElement extends HTMLElement {
  public HTMLFormElement    getForm();
  public String             getAccessKey();
  public void               setAccessKey(String accessKey);
  public boolean            getDisabled();
  public void               setDisabled(boolean disabled);
  public String             getName();
  public void               setName(String name);
  public int                getTabIndex();
  public void               setTabIndex(int tabIndex);
  public String             getType();
  public String             getValue();
  public void               setValue(String value);
}

public interface HTMLLabelElement extends HTMLElement {
  public HTMLFormElement    getForm();
  public String             getAccessKey();
  public void               setAccessKey(String accessKey);
  public String             getHtmlFor();
  public void               setHtmlFor(String htmlFor);
}

public interface HTMLFieldSetElement extends HTMLElement {
  public HTMLFormElement    getForm();
}

public interface HTMLLegendElement extends HTMLElement {
  public HTMLFormElement    getForm();
  public String             getAccessKey();
```

```
  public void                  setAccessKey(String accessKey);
  public String                getAlign();
  public void                  setAlign(String align);
}

public interface HTMLUListElement extends HTMLElement {
  public boolean               getCompact();
  public void                  setCompact(boolean compact);
  public String                getType();
  public void                  setType(String type);
}

public interface HTMLOListElement extends HTMLElement {
  public boolean               getCompact();
  public void                  setCompact(boolean compact);
  public int                   getStart();
  public void                  setStart(int start);
  public String                getType();
  public void                  setType(String type);
}

public interface HTMLDListElement extends HTMLElement {
  public boolean               getCompact();
  public void                  setCompact(boolean compact);
}

public interface HTMLDirectoryElement extends HTMLElement {
  public boolean               getCompact();
  public void                  setCompact(boolean compact);
}

public interface HTMLMenuElement extends HTMLElement {
  public boolean               getCompact();
  public void                  setCompact(boolean compact);
}

public interface HTMLLIElement extends HTMLElement {
  public String                getType();
  public void                  setType(String type);
  public int                   getValue();
  public void                  setValue(int value);
}

public interface HTMLBlockquoteElement extends HTMLElement {
  public String                getCite();
  public void                  setCite(String cite);
}

public interface HTMLDivElement extends HTMLElement {
  public String                getAlign();
  public void                  setAlign(String align);
}

public interface HTMLParagraphElement extends HTMLElement {
  public String                getAlign();
  public void                  setAlign(String align);
}
```

```
public interface HTMLHeadingElement extends HTMLElement {
  public String            getAlign();
  public void              setAlign(String align);
}

public interface HTMLQuoteElement extends HTMLElement {
  public String            getCite();
  public void              setCite(String cite);
}

public interface HTMLPreElement extends HTMLElement {
  public int               getWidth();
  public void              setWidth(int width);
}

public interface HTMLBRElement extends HTMLElement {
  public String            getClear();
  public void              setClear(String clear);
}

public interface HTMLBaseFontElement extends HTMLElement {
  public String            getColor();
  public void              setColor(String color);
  public String            getFace();
  public void              setFace(String face);
  public String            getSize();
  public void              setSize(String size);
}

public interface HTMLFontElement extends HTMLElement {
  public String            getColor();
  public void              setColor(String color);
  public String            getFace();
  public void              setFace(String face);
  public String            getSize();
  public void              setSize(String size);
}

public interface HTMLHRElement extends HTMLElement {
  public String            getAlign();
  public void              setAlign(String align);
  public boolean           getNoShade();
  public void              setNoShade(boolean noShade);
  public String            getSize();
  public void              setSize(String size);
  public String            getWidth();
  public void              setWidth(String width);
}

public interface HTMLModElement extends HTMLElement {
  public String            getCite();
  public void              setCite(String cite);
  public String            getDateTime();
  public void              setDateTime(String dateTime);
}
```

```
public interface HTMLAnchorElement extends HTMLElement {
  public String            getAccessKey();
  public void              setAccessKey(String accessKey);
  public String            getCharset();
  public void              setCharset(String charset);
  public String            getCoords();
  public void              setCoords(String coords);
  public String            getHref();
  public void              setHref(String href);
  public String            getHreflang();
  public void              setHreflang(String hreflang);
  public String            getName();
  public void              setName(String name);
  public String            getRel();
  public void              setRel(String rel);
  public String            getRev();
  public void              setRev(String rev);
  public String            getShape();
  public void              setShape(String shape);
  public int               getTabIndex();
  public void              setTabIndex(int tabIndex);
  public String            getTarget();
  public void              setTarget(String target);
  public String            getType();
  public void              setType(String type);
  public void              blur();
  public void              focus();
}

public interface HTMLImageElement extends HTMLElement {
  public String            getLowSrc();
  public void              setLowSrc(String lowSrc);
  public String            getName();
  public void              setName(String name);
  public String            getAlign();
  public void              setAlign(String align);
  public String            getAlt();
  public void              setAlt(String alt);
  public String            getBorder();
  public void              setBorder(String border);
  public String            getHeight();
  public void              setHeight(String height);
  public String            getHspace();
  public void              setHspace(String hspace);
  public boolean           getIsMap();
  public void              setIsMap(boolean isMap);
  public String            getLongDesc();
  public void              setLongDesc(String longDesc);
  public String            getSrc();
  public void              setSrc(String src);
  public String            getUseMap();
  public void              setUseMap(String useMap);
  public String            getVspace();
  public void              setVspace(String vspace);
  public String            getWidth();
  public void              setWidth(String width);
}
```

```
public interface HTMLObjectElement extends HTMLElement {
  public HTMLFormElement    getForm();
  public String             getCode();
  public void               setCode(String code);
  public String             getAlign();
  public void               setAlign(String align);
  public String             getArchive();
  public void               setArchive(String archive);
  public String             getBorder();
  public void               setBorder(String border);
  public String             getCodeBase();
  public void               setCodeBase(String codeBase);
  public String             getCodeType();
  public void               setCodeType(String codeType);
  public String             getData();
  public void               setData(String data);
  public boolean            getDeclare();
  public void               setDeclare(boolean declare);
  public String             getHeight();
  public void               setHeight(String height);
  public String             getHspace();
  public void               setHspace(String hspace);
  public String             getName();
  public void               setName(String name);
  public String             getStandby();
  public void               setStandby(String standby);
  public int                getTabIndex();
  public void               setTabIndex(int tabIndex);
  public String             getType();
  public void               setType(String type);
  public String             getUseMap();
  public void               setUseMap(String useMap);
  public String             getVspace();
  public void               setVspace(String vspace);
  public String             getWidth();
  public void               setWidth(String width);
}

public interface HTMLParamElement extends HTMLElement {
  public String             getName();
  public void               setName(String name);
  public String             getType();
  public void               setType(String type);
  public String             getValue();
  public void               setValue(String value);
  public String             getValueType();
  public void               setValueType(String valueType);
}

public interface HTMLAppletElement extends HTMLElement {
  public String             getAlign();
  public void               setAlign(String align);
  public String             getAlt();
  public void               setAlt(String alt);
  public String             getArchive();
  public void               setArchive(String archive);
```

```
  public String            getCode();
  public void              setCode(String code);
  public String            getCodeBase();
  public void              setCodeBase(String codeBase);
  public String            getHeight();
  public void              setHeight(String height);
  public String            getHspace();
  public void              setHspace(String hspace);
  public String            getName();
  public void              setName(String name);
  public String            getObject();
  public void              setObject(String object);
  public String            getVspace();
  public void              setVspace(String vspace);
  public String            getWidth();
  public void              setWidth(String width);
}

public interface HTMLMapElement extends HTMLElement {
  public HTMLCollection    getAreas();
  public String            getName();
  public void              setName(String name);
}

public interface HTMLAreaElement extends HTMLElement {
  public String            getAccessKey();
  public void              setAccessKey(String accessKey);
  public String            getAlt();
  public void              setAlt(String alt);
  public String            getCoords();
  public void              setCoords(String coords);
  public String            getHref();
  public void              setHref(String href);
  public boolean           getNoHref();
  public void              setNoHref(boolean noHref);
  public String            getShape();
  public void              setShape(String shape);
  public int               getTabIndex();
  public void              setTabIndex(int tabIndex);
  public String            getTarget();
  public void              setTarget(String target);
}

public interface HTMLScriptElement extends HTMLElement {
  public String            getText();
  public void              setText(String text);
  public String            getHtmlFor();
  public void              setHtmlFor(String htmlFor);
  public String            getEvent();
  public void              setEvent(String event);
  public String            getCharset();
  public void              setCharset(String charset);
  public boolean           getDefer();
  public void              setDefer(boolean defer);
  public String            getSrc();
  public void              setSrc(String src);
  public String            getType();
```

```
  public void                 setType(String type);
}

public interface HTMLTableElement extends HTMLElement {
  public HTMLTableCaptionElement getCaption();
  public void                 setCaption(HTMLTableCaptionElement caption);
  public HTMLTableSectionElement getTHead();
  public void                 setTHead(HTMLTableSectionElement tHead);
  public HTMLTableSectionElement getTFoot();
  public void                 setTFoot(HTMLTableSectionElement tFoot);
  public HTMLCollection       getRows();
  public HTMLCollection       getTBodies();
  public String               getAlign();
  public void                 setAlign(String align);
  public String               getBgColor();
  public void                 setBgColor(String bgColor);
  public String               getBorder();
  public void                 setBorder(String border);
  public String               getCellPadding();
  public void                 setCellPadding(String cellPadding);
  public String               getCellSpacing();
  public void                 setCellSpacing(String cellSpacing);
  public String               getFrame();
  public void                 setFrame(String frame);
  public String               getRules();
  public void                 setRules(String rules);
  public String               getSummary();
  public void                 setSummary(String summary);
  public String               getWidth();
  public void                 setWidth(String width);
  public HTMLElement          createTHead();
  public void                 deleteTHead();
  public HTMLElement          createTFoot();
  public void                 deleteTFoot();
  public HTMLElement          createCaption();
  public void                 deleteCaption();
  public HTMLElement          insertRow(int index);
  public void                 deleteRow(int index);
}

public interface HTMLTableCaptionElement extends HTMLElement {
  public String               getAlign();
  public void                 setAlign(String align);
}

public interface HTMLTableColElement extends HTMLElement {
  public String               getAlign();
  public void                 setAlign(String align);
  public String               getCh();
  public void                 setCh(String ch);
  public String               getChOff();
  public void                 setChOff(String chOff);
  public int                  getSpan();
  public void                 setSpan(int span);
  public String               getVAlign();
  public void                 setVAlign(String vAlign);
  public String               getWidth();
```

```
  public void                  setWidth(String width);
}

public interface HTMLTableSectionElement extends HTMLElement {
  public String                getAlign();
  public void                  setAlign(String align);
  public String                getCh();
  public void                  setCh(String ch);
  public String                getChOff();
  public void                  setChOff(String chOff);
  public String                getVAlign();
  public void                  setVAlign(String vAlign);
  public HTMLCollection        getRows();
  public HTMLElement           insertRow(int index);
  public void                  deleteRow(int index);
}

public interface HTMLTableRowElement extends HTMLElement {
  public int                   getRowIndex();
  public void                  setRowIndex(int rowIndex);
  public int                   getSectionRowIndex();
  public void                  setSectionRowIndex(int sectionRowIndex);
  public HTMLCollection        getCells();
  public void                  setCells(HTMLCollection cells);
  public String                getAlign();
  public void                  setAlign(String align);
  public String                getBgColor();
  public void                  setBgColor(String bgColor);
  public String                getCh();
  public void                  setCh(String ch);
  public String                getChOff();
  public void                  setChOff(String chOff);
  public String                getVAlign();
  public void                  setVAlign(String vAlign);
  public HTMLElement           insertCell(int index);
  public void                  deleteCell(int index);
}

public interface HTMLTableCellElement extends HTMLElement {
  public int                   getCellIndex();
  public void                  setCellIndex(int cellIndex);
  public String                getAbbr();
  public void                  setAbbr(String abbr);
  public String                getAlign();
  public void                  setAlign(String align);
  public String                getAxis();
  public void                  setAxis(String axis);
  public String                getBgColor();
  public void                  setBgColor(String bgColor);
  public String                getCh();
  public void                  setCh(String ch);
  public String                getChOff();
  public void                  setChOff(String chOff);
  public int                   getColSpan();
  public void                  setColSpan(int colSpan);
  public String                getHeaders();
  public void                  setHeaders(String headers);
```

```
  public String               getHeight();
  public void                 setHeight(String height);
  public boolean              getNoWrap();
  public void                 setNoWrap(boolean noWrap);
  public int                  getRowSpan();
  public void                 setRowSpan(int rowSpan);
  public String               getScope();
  public void                 setScope(String scope);
  public String               getVAlign();
  public void                 setVAlign(String vAlign);
  public String               getWidth();
  public void                 setWidth(String width);
}

public interface HTMLFrameSetElement extends HTMLElement {
  public String               getCols();
  public void                 setCols(String cols);
  public String               getRows();
  public void                 setRows(String rows);
}

public interface HTMLFrameElement extends HTMLElement {
  public String               getFrameBorder();
  public void                 setFrameBorder(String frameBorder);
  public String               getLongDesc();
  public void                 setLongDesc(String longDesc);
  public String               getMarginHeight();
  public void                 setMarginHeight(String marginHeight);
  public String               getMarginWidth();
  public void                 setMarginWidth(String marginWidth);
  public String               getName();
  public void                 setName(String name);
  public boolean              getNoResize();
  public void                 setNoResize(boolean noResize);
  public String               getScrolling();
  public void                 setScrolling(String scrolling);
  public String               getSrc();
  public void                 setSrc(String src);
}

public interface HTMLIFrameElement extends HTMLElement {
  public String               getAlign();
  public void                 setAlign(String align);
  public String               getFrameBorder();
  public void                 setFrameBorder(String frameBorder);
  public String               getHeight();
  public void                 setHeight(String height);
  public String               getLongDesc();
  public void                 setLongDesc(String longDesc);
  public String               getMarginHeight();
  public void                 setMarginHeight(String marginHeight);
  public String               getMarginWidth();
  public void                 setMarginWidth(String marginWidth);
  public String               getName();
  public void                 setName(String name);
  public String               getScrolling();
  public void                 setScrolling(String scrolling);
```

```
  public String              getSrc();
  public void                setSrc(String src);
  public String              getWidth();
  public void                setWidth(String width);
}
```

Appendix E: ECMA Script Language Binding

This appendix contains the complete ECMA Script binding for the Level 1 Document Object Model definitions. The definitions are divided into Core and HTML.

E.1: Document Object Model Level 1 Core

Object **DOMException**
Object **ExceptionCode**
Object **DOMImplementation**
The **DOMImplementation** object has the following methods:
hasFeature(feature, version)
This method returns a **boolean**. The **feature** parameter is of type **DOMString**. The **version** parameter is of type **DOMString**.
Object **DocumentFragment**
DocumentFragment has the all the properties and methods of **Node** as well as the properties and methods defined below.
Object **Document**
Document has the all the properties and methods of **Node** as well as the properties and methods defined below.
The **Document** object has the following properties:
doctype
This property is of type **DocumentType**.
implementation
This property is of type **DOMImplementation**.
documentElement
This property is of type **Element**.
The **Document** object has the following methods:
createElement(tagName)
This method returns a **Element**. The **tagName** parameter is of type **DOMString**.
createDocumentFragment()
This method returns a **DocumentFragment**.
createTextNode(data)
This method returns a **Text**. The **data** parameter is of type **DOMString**.
createComment(data)
This method returns a **Comment**. The **data** parameter is of type **DOMString**.
createCDATASection(data)
This method returns a **CDATASection**. The **data** parameter is of type **DOMString**.
createProcessingInstruction(target, data)
This method returns a **ProcessingInstruction**. The **target** parameter is of type **DOMString**. The **data** parameter is of type **DOMString**.
createAttribute(name)
This method returns a **Attr**. The **name** parameter is of type **DOMString**.
createEntityReference(name)
This method returns a **EntityReference**. The **name** parameter is of type **DOMString**.

getElementsByTagName(tagname)
This method returns a **NodeList**. The **tagname** parameter is of type **DOMString**.

Object **Node**

The **Node** object has the following properties:

nodeName
This property is of type **String**.

nodeValue
This property is of type **String**.

nodeType
This property is of type **short**.

parentNode
This property is of type **Node**.

childNodes
This property is of type **NodeList**.

firstChild
This property is of type **Node**.

lastChild
This property is of type **Node**.

previousSibling
This property is of type **Node**.

nextSibling
This property is of type **Node**.

attributes
This property is of type **NamedNodeMap**.

ownerDocument
This property is of type **Document**.

The **Node** object has the following methods:

insertBefore(newChild, refChild)
This method returns a **Node**. The **newChild** parameter is of type **Node**. The **refChild** parameter is of type **Node**.

replaceChild(newChild, oldChild)
This method returns a **Node**. The **newChild** parameter is of type **Node**. The **oldChild** parameter is of type **Node**.

removeChild(oldChild)
This method returns a **Node**. The **oldChild** parameter is of type **Node**.

appendChild(newChild)
This method returns a **Node**. The **newChild** parameter is of type **Node**.

hasChildNodes()
This method returns a **boolean**.

cloneNode(deep)
This method returns a **Node**. The **deep** parameter is of type **boolean**.

Object **NodeList**

The **NodeList** object has the following properties:

length
This property is of type **int**.

The **NodeList** object has the following methods:

item(index)

This method returns a **Node**. The **index** parameter is of type **unsigned long**.

Object **NamedNodeMap**

The **NamedNodeMap** object has the following properties:

length

This property is of type **int**.

The **NamedNodeMap** object has the following methods:

getNamedItem(name)

This method returns a **Node**. The **name** parameter is of type **DOMString**.

setNamedItem(arg)

This method returns a **Node**. The **arg** parameter is of type **Node**.

removeNamedItem(name)

This method returns a **Node**. The **name** parameter is of type **DOMString**.

item(index)

This method returns a **Node**. The **index** parameter is of type **unsigned long**.

Object **CharacterData**

CharacterData has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **CharacterData** object has the following properties:

data

This property is of type **String**.

length

This property is of type **int**.

The **CharacterData** object has the following methods:

substringData(offset, count)

This method returns a **DOMString**. The **offset** parameter is of type **unsigned long**. The **count** parameter is of type **unsigned long**.

appendData(arg)

This method returns a **void**. The **arg** parameter is of type **DOMString**.

insertData(offset, arg)

This method returns a **void**. The **offset** parameter is of type **unsigned long**. The **arg** parameter is of type **DOMString**.

deleteData(offset, count)

This method returns a **void**. The **offset** parameter is of type **unsigned long**. The **count** parameter is of type **unsigned long**.

replaceData(offset, count, arg)

This method returns a **void**. The **offset** parameter is of type **unsigned long**. The **count** parameter is of type **unsigned long**. The **arg** parameter is of type **DOMString**.

Object **Attr**

Attr has the all the properties and methods of **Node** as well as the properties and methods defined below.

The **Attr** object has the following properties:

name

This property is of type **String**.

specified
This property is of type **boolean**.
value
This property is of type **String**.

Object **Element**
Element has the all the properties and methods of **Node** as well as the properties and methods defined below.
The **Element** object has the following properties:
tagName
This property is of type **String**.
The **Element** object has the following methods:
getAttribute(name)
This method returns a **DOMString**. The **name** parameter is of type **DOMString**.
setAttribute(name, value)
This method returns a **void**. The **name** parameter is of type **DOMString**. The **value** parameter is of type **DOMString**.
removeAttribute(name)
This method returns a **void**. The **name** parameter is of type **DOMString**.
getAttributeNode(name)
This method returns a **Attr**. The **name** parameter is of type **DOMString**.
setAttributeNode(newAttr)
This method returns a **Attr**. The **newAttr** parameter is of type **Attr**.
removeAttributeNode(oldAttr)
This method returns a **Attr**. The **oldAttr** parameter is of type **Attr**.
getElementsByTagName(name)
This method returns a **NodeList**. The **name** parameter is of type **DOMString**.
normalize()
This method returns a **void**.

Object **Text**
Text has the all the properties and methods of **CharacterData** as well as the properties and methods defined below.
The **Text** object has the following methods:
splitText(offset)
This method returns a **Text**. The **offset** parameter is of type **unsigned long**.

Object **Comment**
Comment has the all the properties and methods of **CharacterData** as well as the properties and methods defined below.

Object **CDATASection**
CDATASection has the all the properties and methods of **Text** as well as the properties and methods defined below.

Object **DocumentType**
DocumentType has the all the properties and methods of **Node** as well as the properties and methods defined below.
The **DocumentType** object has the following properties:
name
This property is of type **String**.

entities
This property is of type **NamedNodeMap**.
notations
This property is of type **NamedNodeMap**.

Object **Notation**
Notation has the all the properties and methods of **Node** as well as the properties and methods defined below.
The **Notation** object has the following properties:
publicId
This property is of type **String**.
systemId
This property is of type **String**.

Object **Entity**
Entity has the all the properties and methods of **Node** as well as the properties and methods defined below.
The **Entity** object has the following properties:
publicId
This property is of type **String**.
systemId
This property is of type **String**.
notationName
This property is of type **String**.

Object **EntityReference**
EntityReference has the all the properties and methods of **Node** as well as the properties and methods defined below.

Object **ProcessingInstruction**
ProcessingInstruction has the all the properties and methods of **Node** as well as the properties and methods defined below.
The **ProcessingInstruction** object has the following properties:
target
This property is of type **String**.
data
This property is of type **String**.

E.2: Document Object Model Level 1 HTML

Object **HTMLCollection**
The **HTMLCollection** object has the following properties:
length
This property is of type **int**.
The **HTMLCollection** object has the following methods:
item(index)
This method returns a **Node**. The **index** parameter is of type **unsigned long**.
namedItem(name)
This method returns a **Node**. The **name** parameter is of type **DOMString**.

Object **HTMLDocument**

HTMLDocument has the all the properties and methods of **Document** as well as the properties and methods defined below.

The **HTMLDocument** object has the following properties:

title

This property is of type **String.**

referrer

This property is of type **String.**

domain

This property is of type **String.**

URL

This property is of type **String.**

body

This property is of type **HTMLElement.**

images

This property is of type **HTMLCollection.**

applets

This property is of type **HTMLCollection.**

links

This property is of type **HTMLCollection.**

forms

This property is of type **HTMLCollection.**

anchors

This property is of type **HTMLCollection.**

cookie

This property is of type **String.**

The **HTMLDocument** object has the following methods:

open()

This method returns a **void.**

close()

This method returns a **void.**

write(text)

This method returns a **void.** The **text** parameter is of type **DOMString.**

writeln(text)

This method returns a **void.** The **text** parameter is of type **DOMString.**

getElementById(elementId)

This method returns a **Element.** The **elementId** parameter is of type **DOMString.**

getElementsByName(elementName)

This method returns a **NodeList.** The **elementName** parameter is of type **DOMString.**

Object **HTMLElement**

HTMLElement has the all the properties and methods of **Element** as well as the properties and methods defined below.

The **HTMLElement** object has the following properties:

id

This property is of type **String.**

title
This property is of type **String**.
lang
This property is of type **String**.
dir
This property is of type **String**.
className
This property is of type **String**.

Object **HTMLHtmlElement**

HTMLHtmlElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLHtmlElement** object has the following properties:

version
This property is of type **String**.

Object **HTMLHeadElement**

HTMLHeadElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLHeadElement** object has the following properties:

profile
This property is of type **String**.

Object **HTMLLinkElement**

HTMLLinkElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLLinkElement** object has the following properties:

disabled
This property is of type **boolean**.
charset
This property is of type **String**.
href
This property is of type **String**.
hreflang
This property is of type **String**.
media
This property is of type **String**.
rel
This property is of type **String**.
rev
This property is of type **String**.
target
This property is of type **String**.
type
This property is of type **String**.

Object **HTMLTitleElement**

HTMLTitleElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTitleElement** object has the following properties:

text

This property is of type **String.**

Object **HTMLMetaElement**

HTMLMetaElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLMetaElement** object has the following properties:

content

This property is of type **String.**

httpEquiv

This property is of type **String.**

name

This property is of type **String.**

scheme

This property is of type **String.**

Object **HTMLBaseElement**

HTMLBaseElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLBaseElement** object has the following properties:

href

This property is of type **String.**

target

This property is of type **String.**

Object **HTMLIsIndexElement**

HTMLIsIndexElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLIsIndexElement** object has the following properties:

form

This property is of type **HTMLFormElement.**

prompt

This property is of type **String.**

Object **HTMLStyleElement**

HTMLStyleElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLStyleElement** object has the following properties:

disabled

This property is of type **boolean.**

media

This property is of type **String.**

type

This property is of type **String.**

Object **HTMLBodyElement**

HTMLBodyElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLBodyElement** object has the following properties:

aLink
This property is of type **String**.
background
This property is of type **String**.
bgColor
This property is of type **String**.
link
This property is of type **String**.
text
This property is of type **String**.
vLink
This property is of type **String**.

Object **HTMLFormElement**

HTMLFormElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLFormElement** object has the following properties:

elements
This property is of type **HTMLCollection**.
length
This property is of type **long**.
name
This property is of type **String**.
acceptCharset
This property is of type **String**.
action
This property is of type **String**.
enctype
This property is of type **String**.
method
This property is of type **String**.
target
This property is of type **String**.

The **HTMLFormElement** object has the following methods:

submit()
This method returns a **void**.
reset()
This method returns a **void**.

Object **HTMLSelectElement**

HTMLSelectElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLSelectElement** object has the following properties:

type
This property is of type **String**.
selectedIndex
This property is of type **long**.

value
This property is of type **String.**
length
This property is of type **long.**
form
This property is of type **HTMLFormElement.**
options
This property is of type **HTMLCollection.**
disabled
This property is of type **boolean.**
multiple
This property is of type **boolean.**
name
This property is of type **String.**
size
This property is of type **long.**
tabIndex
This property is of type **long.**

The **HTMLSelectElement** object has the following methods:
add(element, before)
This method returns a **void.** The **element** parameter is of type **HTMLElement.** The **before** parameter is of type **HTMLElement.**
remove(index)
This method returns a **void.** The **index** parameter is of type **long.**
blur()
This method returns a **void.**
focus()
This method returns a **void.**

Object **HTMLOptGroupElement**

HTMLOptGroupElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLOptGroupElement** object has the following properties:
disabled
This property is of type **boolean.**
label
This property is of type **String.**

Object **HTMLOptionElement**

HTMLOptionElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLOptionElement** object has the following properties:
form
This property is of type **HTMLFormElement.**
defaultSelected
This property is of type **boolean.**
text
This property is of type **String.**

index
This property is of type **long.**
disabled
This property is of type **boolean.**
label
This property is of type **String.**
selected
This property is of type **boolean.**
value
This property is of type **String.**

Object **HTMLInputElement**

HTMLInputElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLInputElement** object has the following properties:

defaultValue
This property is of type **String.**
defaultChecked
This property is of type **boolean.**
form
This property is of type **HTMLFormElement.**
accept
This property is of type **String.**
accessKey
This property is of type **String.**
align
This property is of type **String.**
alt
This property is of type **String.**
checked
This property is of type **boolean.**
disabled
This property is of type **boolean.**
maxLength
This property is of type **long.**
name
This property is of type **String.**
readOnly
This property is of type **boolean.**
size
This property is of type **String.**
src
This property is of type **String.**
tabIndex
This property is of type **long.**
type
This property is of type **String.**

useMap
This property is of type **String**.
value
This property is of type **String**.

The **HTMLInputElement** object has the following methods:
blur()
This method returns a **void**.
focus()
This method returns a **void**.
select()
This method returns a **void**.
click()
This method returns a **void**.

Object **HTMLTextAreaElement**

HTMLTextAreaElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTextAreaElement** object has the following properties:
defaultValue
This property is of type **String**.
form
This property is of type **HTMLFormElement**.
accessKey
This property is of type **String**.
cols
This property is of type **long**.
disabled
This property is of type **boolean**.
name
This property is of type **String**.
readOnly
This property is of type **boolean**.
rows
This property is of type **long**.
tabIndex
This property is of type **long**.
type
This property is of type **String**.
value
This property is of type **String**.

The **HTMLTextAreaElement** object has the following methods:
blur()
This method returns a **void**.
focus()
This method returns a **void**.
select()
This method returns a **void**.

Object **HTMLButtonElement**

HTMLButtonElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLButtonElement** object has the following properties:

form

This property is of type **HTMLFormElement.**

accessKey

This property is of type **String.**

disabled

This property is of type **boolean.**

name

This property is of type **String.**

tabIndex

This property is of type **long.**

type

This property is of type **String.**

value

This property is of type **String.**

Object **HTMLLabelElement**

HTMLLabelElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLLabelElement** object has the following properties:

form

This property is of type **HTMLFormElement.**

accessKey

This property is of type **String.**

htmlFor

This property is of type **String.**

Object **HTMLFieldSetElement**

HTMLFieldSetElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLFieldSetElement** object has the following properties:

form

This property is of type **HTMLFormElement.**

Object **HTMLLegendElement**

HTMLLegendElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLLegendElement** object has the following properties:

form

This property is of type **HTMLFormElement.**

accessKey

This property is of type **String.**

align

This property is of type **String.**

Object **HTMLUListElement**

HTMLUListElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLUListElement** object has the following properties:

compact

This property is of type **boolean.**

type

This property is of type **String.**

Object **HTMLOListElement**

HTMLOListElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLOListElement** object has the following properties:

compact

This property is of type **boolean.**

start

This property is of type **long.**

type

This property is of type **String.**

Object **HTMLDListElement**

HTMLDListElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLDListElement** object has the following properties:

compact

This property is of type **boolean.**

Object **HTMLDirectoryElement**

HTMLDirectoryElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLDirectoryElement** object has the following properties:

compact

This property is of type **boolean.**

Object **HTMLMenuElement**

HTMLMenuElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLMenuElement** object has the following properties:

compact

This property is of type **boolean.**

Object **HTMLLIElement**

HTMLLIElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLLIElement** object has the following properties:

type

This property is of type **String.**

value

This property is of type **long.**

Object **HTMLBlockquoteElement**

HTMLBlockquoteElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLBlockquoteElement** object has the following properties:

cite

This property is of type **String**.

Object **HTMLDivElement**

HTMLDivElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLDivElement** object has the following properties:

align

This property is of type **String**.

Object **HTMLParagraphElement**

HTMLParagraphElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLParagraphElement** object has the following properties:

align

This property is of type **String**.

Object **HTMLHeadingElement**

HTMLHeadingElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLHeadingElement** object has the following properties:

align

This property is of type **String**.

Object **HTMLQuoteElement**

HTMLQuoteElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLQuoteElement** object has the following properties:

cite

This property is of type **String**.

Object **HTMLPreElement**

HTMLPreElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLPreElement** object has the following properties:

width

This property is of type **long**.

Object **HTMLBRElement**

HTMLBRElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLBRElement** object has the following properties:

clear

This property is of type **String**.

Object **HTMLBaseFontElement**

HTMLBaseFontElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLBaseFontElement** object has the following properties:

color

This property is of type **String**.

face

This property is of type **String.**

size

This property is of type **String.**

Object **HTMLFontElement**

HTMLFontElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLFontElement** object has the following properties:

color

This property is of type **String.**

face

This property is of type **String.**

size

This property is of type **String.**

Object **HTMLHRElement**

HTMLHRElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLHRElement** object has the following properties:

align

This property is of type **String.**

noShade

This property is of type **boolean.**

size

This property is of type **String.**

width

This property is of type **String.**

Object **HTMLModElement**

HTMLModElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLModElement** object has the following properties:

cite

This property is of type **String.**

dateTime

This property is of type **String.**

Object **HTMLAnchorElement**

HTMLAnchorElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLAnchorElement** object has the following properties:

accessKey

This property is of type **String.**

charset

This property is of type **String.**

coords

This property is of type **String.**

href

This property is of type **String.**

hreflang

This property is of type **String**.

name

This property is of type **String**.

rel

This property is of type **String**.

rev

This property is of type **String**.

shape

This property is of type **String**.

tabIndex

This property is of type **long**.

target

This property is of type **String**.

type

This property is of type **String**.

The **HTMLAnchorElement** object has the following methods:

blur()

This method returns a **void**.

focus()

This method returns a **void**.

Object **HTMLImageElement**

HTMLImageElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLImageElement** object has the following properties:

lowSrc

This property is of type **String**.

name

This property is of type **String**.

align

This property is of type **String**.

alt

This property is of type **String**.

border

This property is of type **String**.

height

This property is of type **String**.

hspace

This property is of type **String**.

isMap

This property is of type **boolean**.

longDesc

This property is of type **String**.

src

This property is of type **String**.

useMap
This property is of type **String**.
vspace
This property is of type **String**.
width
This property is of type **String**.

Object **HTMLObjectElement**

HTMLObjectElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLObjectElement** object has the following properties:

form
This property is of type **HTMLFormElement**.
code
This property is of type **String**.
align
This property is of type **String**.
archive
This property is of type **String**.
border
This property is of type **String**.
codeBase
This property is of type **String**.
codeType
This property is of type **String**.
data
This property is of type **String**.
declare
This property is of type **boolean**.
height
This property is of type **String**.
hspace
This property is of type **String**.
name
This property is of type **String**.
standby
This property is of type **String**.
tabIndex
This property is of type **long**.
type
This property is of type **String**.
useMap
This property is of type **String**.
vspace
This property is of type **String**.
width
This property is of type **String**.

Object **HTMLParamElement**

HTMLParamElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLParamElement** object has the following properties:

name
This property is of type **String**.

type
This property is of type **String**.

value
This property is of type **String**.

valueType
This property is of type **String**.

Object **HTMLAppletElement**

HTMLAppletElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLAppletElement** object has the following properties:

align
This property is of type **String**.

alt
This property is of type **String**.

archive
This property is of type **String**.

code
This property is of type **String**.

codeBase
This property is of type **String**.

height
This property is of type **String**.

hspace
This property is of type **String**.

name
This property is of type **String**.

object
This property is of type **String**.

vspace
This property is of type **String**.

width
This property is of type **String**.

Object **HTMLMapElement**

HTMLMapElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLMapElement** object has the following properties:

areas
This property is of type **HTMLCollection**.

name
This property is of type **String**.

Object **HTMLAreaElement**

HTMLAreaElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLAreaElement** object has the following properties:

accessKey
This property is of type **String**.

alt
This property is of type **String**.

coords
This property is of type **String**.

href
This property is of type **String**.

noHref
This property is of type **boolean**.

shape
This property is of type **String**.

tabIndex
This property is of type **long**.

target
This property is of type **String**.

Object **HTMLScriptElement**

HTMLScriptElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLScriptElement** object has the following properties:

text
This property is of type **String**.

htmlFor
This property is of type **String**.

event
This property is of type **String**.

charset
This property is of type **String**.

defer
This property is of type **boolean**.

src
This property is of type **String**.

type
This property is of type **String**.

Object **HTMLTableElement**

HTMLTableElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableElement** object has the following properties:

caption
This property is of type **HTMLTableCaptionElement**.

tHead
This property is of type **HTMLTableSectionElement**.

tFoot
This property is of type **HTMLTableSectionElement.**
rows
This property is of type **HTMLCollection.**
tBodies
This property is of type **HTMLCollection.**
align
This property is of type **String.**
bgColor
This property is of type **String.**
border
This property is of type **String.**
cellPadding
This property is of type **String.**
cellSpacing
This property is of type **String.**
frame
This property is of type **String.**
rules
This property is of type **String.**
summary
This property is of type **String.**
width
This property is of type **String.**

The **HTMLTableElement** object has the following methods:

createTHead()
This method returns a **HTMLElement.**
deleteTHead()
This method returns a **void.**
createTFoot()
This method returns a **HTMLElement.**
deleteTFoot()
This method returns a **void.**
createCaption()
This method returns a **HTMLElement.**
deleteCaption()
This method returns a **void.**
insertRow(index)
This method returns a **HTMLElement**. The **index** parameter is of type **long.**
deleteRow(index)
This method returns a **void**. The **index** parameter is of type **long.**

Object **HTMLTableCaptionElement**

HTMLTableCaptionElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableCaptionElement** object has the following properties:

align

This property is of type **String**.

Object **HTMLTableColElement**

HTMLTableColElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableColElement** object has the following properties:

align

This property is of type **String**.

ch

This property is of type **String**.

chOff

This property is of type **String**.

span

This property is of type **long**.

vAlign

This property is of type **String**.

width

This property is of type **String**.

Object **HTMLTableSectionElement**

HTMLTableSectionElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableSectionElement** object has the following properties:

align

This property is of type **String**.

ch

This property is of type **String**.

chOff

This property is of type **String**.

vAlign

This property is of type **String**.

rows

This property is of type **HTMLCollection**.

The **HTMLTableSectionElement** object has the following methods:

insertRow(index)

This method returns a **HTMLElement.** The **index** parameter is of type **long**.

deleteRow(index)

This method returns a **void**. The **index** parameter is of type **long**.

Object **HTMLTableRowElement**

HTMLTableRowElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableRowElement** object has the following properties:

rowIndex

This property is of type **long**.

sectionRowIndex

This property is of type **long**.

cells
This property is of type **HTMLCollection.**
align
This property is of type **String**.
bgColor
This property is of type **String**.
ch
This property is of type **String**.
chOff
This property is of type **String**.
vAlign
This property is of type **String**.

The **HTMLTableRowElement** object has the following methods:
insertCell(index)
This method returns a **HTMLElement**. The **index** parameter is of type **long**.
deleteCell(index)
This method returns a **void**. The **index** parameter is of type **long**.

Object **HTMLTableCellElement**

HTMLTableCellElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.

The **HTMLTableCellElement** object has the following properties:
cellIndex
This property is of type **long**.
abbr
This property is of type **String**.
align
This property is of type **String**.
axis
This property is of type **String**.
bgColor
This property is of type **String**.
ch
This property is of type **String**.
chOff
This property is of type **String**.
colSpan
This property is of type **long**.
headers
This property is of type **String**.
height
This property is of type **String**.
noWrap
This property is of type **boolean**.
rowSpan
This property is of type **long**.

scope
This property is of type **String.**
vAlign
This property is of type **String.**
width
This property is of type **String.**

Object **HTMLFrameSetElement**
HTMLFrameSetElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.
The **HTMLFrameSetElement** object has the following properties:
cols
This property is of type **String.**
rows
This property is of type **String.**

Object **HTMLFrameElement**
HTMLFrameElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.
The **HTMLFrameElement** object has the following properties:
frameBorder
This property is of type **String.**
longDesc
This property is of type **String.**
marginHeight
This property is of type **String.**
marginWidth
This property is of type **String.**
name
This property is of type **String.**
noResize
This property is of type **boolean.**
scrolling
This property is of type **String.**
src
This property is of type **String.**

Object **HTMLIFrameElement**
HTMLIFrameElement has the all the properties and methods of **HTMLElement** as well as the properties and methods defined below.
The **HTMLIFrameElement** object has the following properties:
align
This property is of type **String.**
frameBorder
This property is of type **String.**
height
This property is of type **String.**
longDesc
This property is of type **String.**

marginHeight
This property is of type **String**.

marginWidth
This property is of type **String**.

name
This property is of type **String**.

scrolling
This property is of type **String**.

src
This property is of type **String**.

width
This property is of type **String**.

References

XML

W3C (World Wide Web Consortium) *Extensible Markup Language (XML) 1.0*. See http://www.w3.org/TR/REC-xml .

HTML4.0

W3C (World Wide Web Consortium) *HTML 4.0 Specification*. See http://www.w3.org/TR/REC-html40 .

Unicode

The Unicode Consortium. *The Unicode Standard, Version* 2.0. Reading, Mass.: Addison-Wesley Developers Press, 1996.

CORBA

OMG (Object Management Group) *The Common Object Request Broker: Architecture and Specification*. See http://www.omg.org/corba/corbiiop.htm .

Java

Sun *The Java Language Specification*. See http://java.sun.com/docs/books/jls/ .

ECMAScript

ECMA (European Computer Manufacturers Association) *ECMAScript Language Specification*. See http://www.ecma.ch/stand/ECMA-262.htm .

Index

Production Notes (Non-Normative)

Editors
Gavin Nicol, Inso EPS

The DOM specification serves as a good example of the power of using XML: all of the HTML documents, Java bindings, OMG IDL bindings, and ECMA Script bindings are generated from a single set of XML source files. This section outlines how this specification is written in XML, and how the various derived works are created.

1. The Document Type Definition

This specification was written entirely in XML, using a DTD based heavily on the DTD used by the XML Working Group for the XML specification. The major difference between the DTD used by the XML Working Group, and the DTD used for this specification is the addition of a DTD module for interface specifications.

The DTD module for interfaces specifications is a very loose translation of the Extended Backus-Naur Form (EBNF) specification of the OMG IDL syntax into XML DTD syntax. In addition to the translation, the ability to *describe* the interfaces was added, thereby creating a limited form of *literate programming* for interface definitions.

While the DTD module is sufficient for the purposes of the DOM WG, it is very loosely typed, meaning that there are very few constraints placed on the type specifications (the type information is effectively treated as an opaque string). In a DTD for object to object communication, some stricter enforcement of data types would probably be beneficial.

2. The production process

The DOM specification is written using XML. All documents are valid XML. In order to produce the HTML versions of the specification, the object indexes, the Java source code, and the OMG IDL and ECMA Script definitions, the XML specification is *converted*.

The tool currently used for conversion is *COST* by Joe English. `COST` takes the ESIS output of `nsgmls`, creates an internal representation, and then allows *scripts*, and *event handlers* to be run over the internal data structure. Event handlers allow document *patterns* and associated processing to be specified: when the pattern is matched during a pre-order traversal of a document subtree, the associated action is executed. This is the heart of the conversion process. Scripts are used to tie the various components together. For example, each of the major derived data sources (Java code etc.) is created by the execution of a script, which in turn executes one or more event handlers. The scripts and event handlers are specified using TCL.

The current version of `COST` has been somewhat modified from the publicly available version. In particular, it now runs correctly under 32-bit Windows, uses TCL 8.0, and correctly handles the case sensitivity of XML (though it probably could not correctly handle native language markup).

We could also have used `Jade`, by James Clark. Like `COST`, `Jade` allows patterns and actions to be specified, but `Jade` is based on DSSSL, an international standard, whereas `COST` is not. `Jade` is more powerful than `COST` in many ways, but prior experience of the editor with Cost made it easier to use this rather than `Jade`. A future version or Level of the DOM specification may be produced using `Jade` or an `XSL` processor.

The complete XML source files are available at:
http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/xml-source.zip

3. Object Definitions

As stated earlier, all object definitions are specified in XML. The Java bindings, OMG IDL bindings, and ECMA Script bindings are all generated automatically from the XML source code.

This is possible because the information specified in XML is a *superset* of what these other syntax need. This is a general observation, and the same kind of technique can be applied to many other areas: given rich structure, rich processing and conversion are possible. For Java and OMG IDL, it is basically just a matter of renaming syntactic keywords; for ECMA Script, the process is somewhat more involved.

A typical object definition in XML looks something like this:

```
<interface name="foo">
  <descr><p>Description goes here...</p></descr>
  <method name="bar">
    <descr><p>Description goes here...</p></descr>
    <parameters>
      <param name="baz" type="DOMString" attr="in">
        <descr><p>Description goes here...</p></descr>
      </param>
    </parameters>
    <returns type="void">
       <descr><p>Description goes here...</p></descr>
    </returns>
    <raises>
       <!-- Throws no exceptions -->
    </raises>
  </method>
</interface>
```

As can easily be seen, this is quite verbose, but not unlike OMG IDL. In fact, when the specification was originally converted to use XML, the OMG IDL definitions were automatically converted into the corresponding XML source using common Unix text manipulation tools.

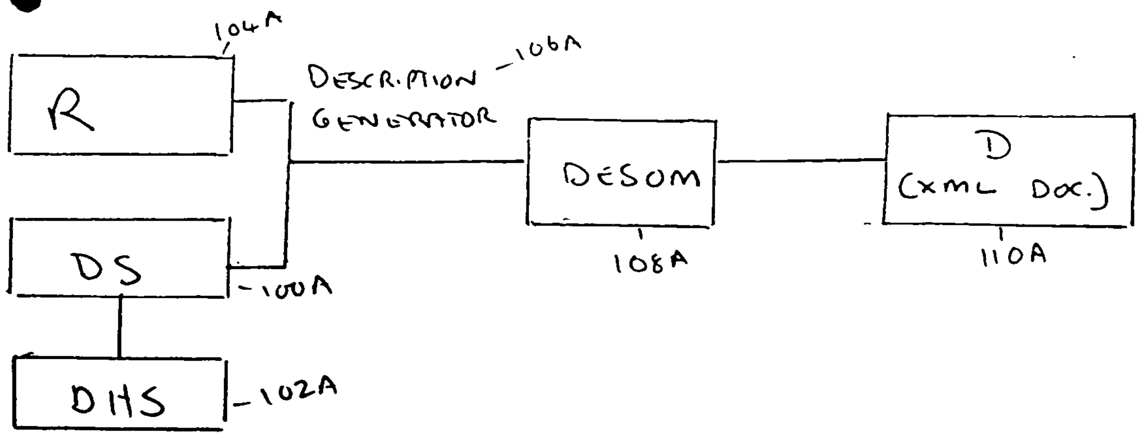

104A
R
DESCRIPTION GENERATOR
106A
DESOM
D (XML DOC.)
DS
100A
108A
110A
DHS
102A

Fig. 1A

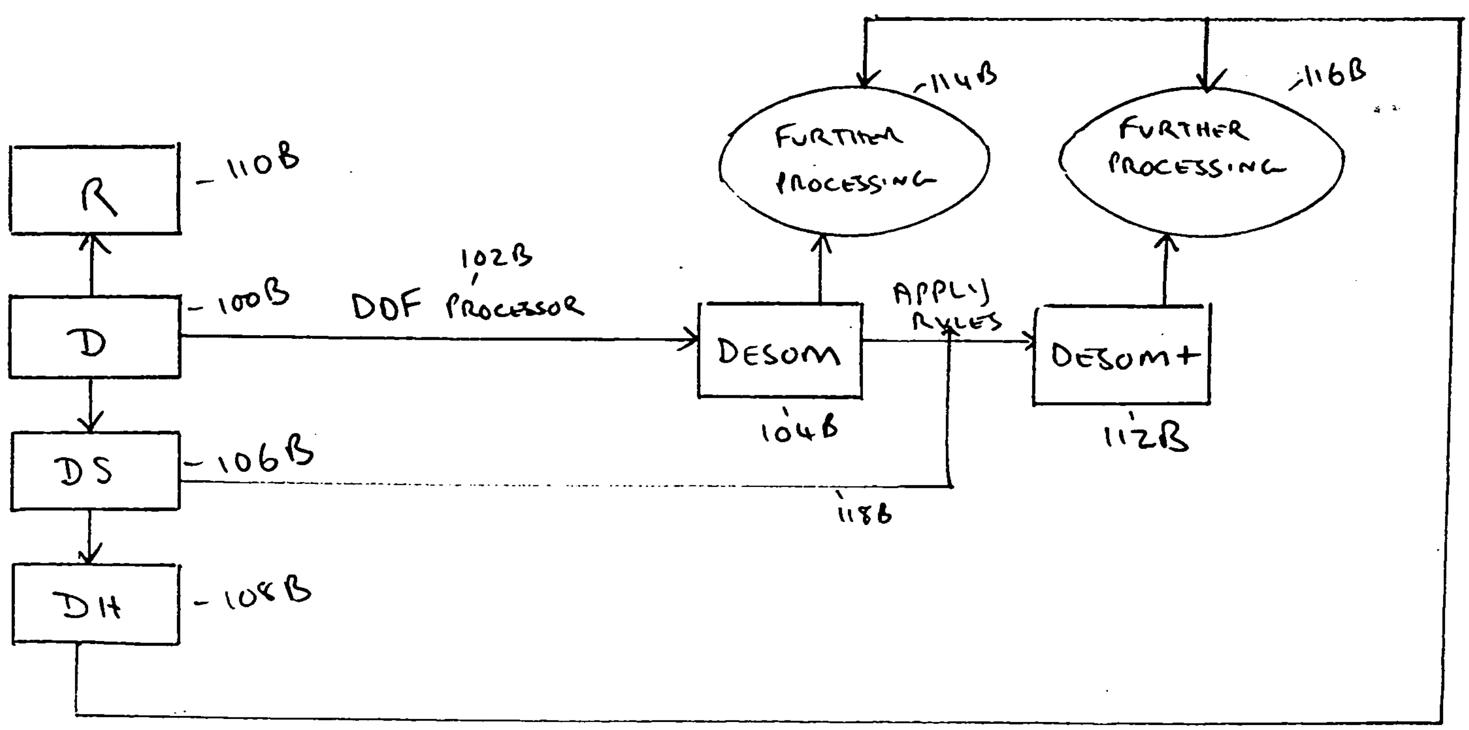

114B
116B
FURTHER PROCESSING
FURTHER PROCESSING
R
110B
102B
D
100B
DDF PROCESSOR
DESOM
APPLY RULES
DESOM+
104B
112B
DS
106B
118B
DH
108B

Fig. 1B

2/16

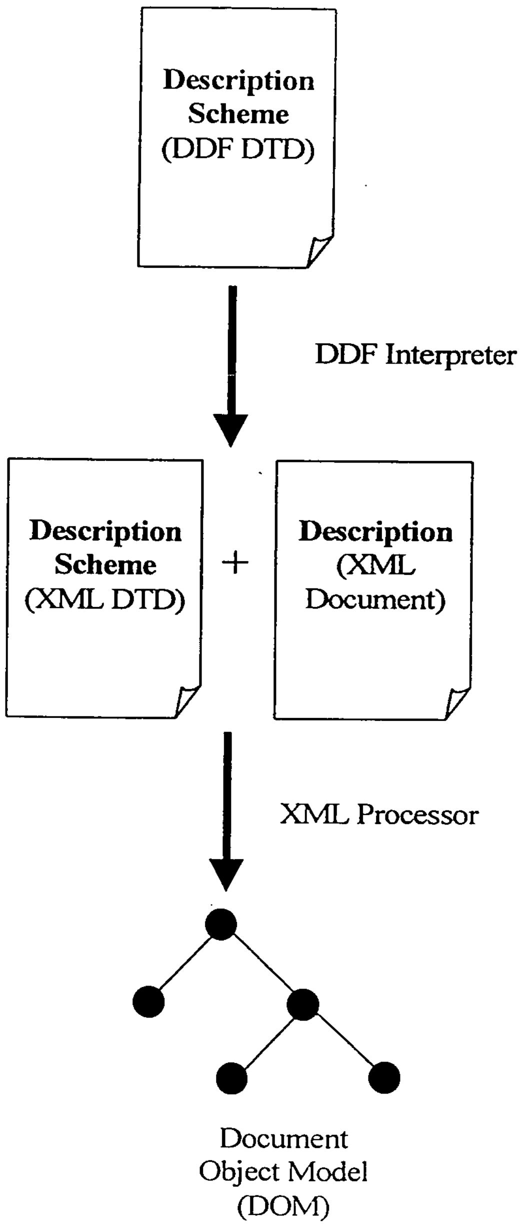


Fig. 2A

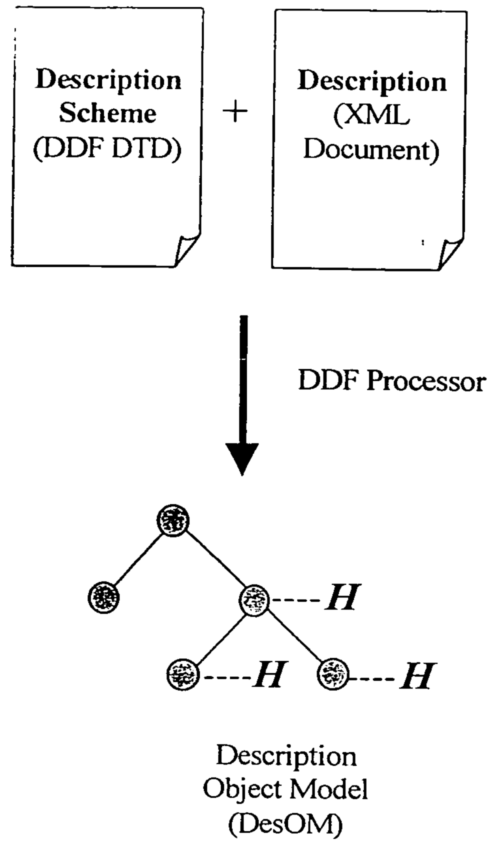


Fig. 2B

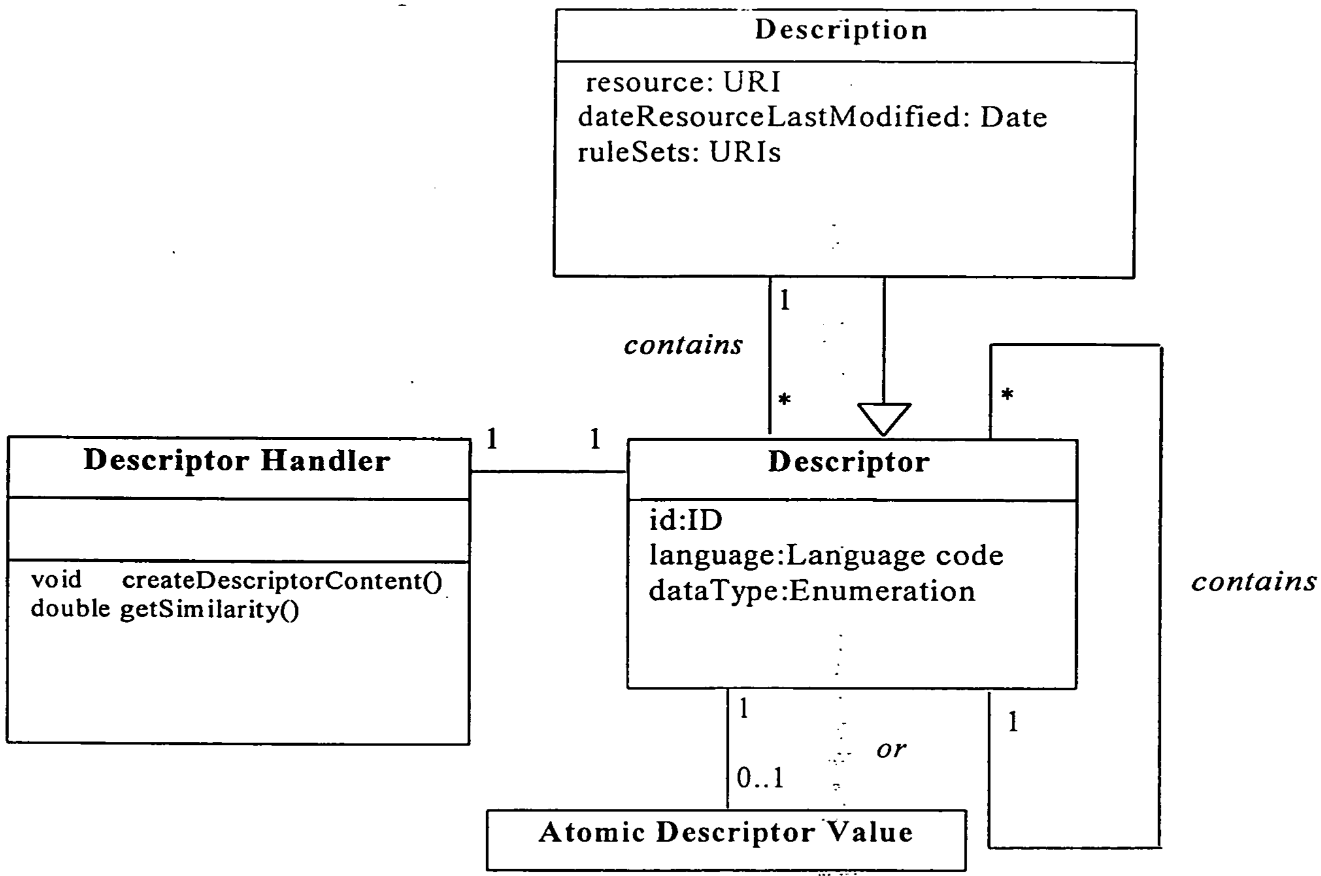

Description
resource: URI
dateResourceLastModified: Date
ruleSets: URIs
1
contains
*
*
Descriptor Handler
1
1
Descriptor
void createDescriptorContent()
double getSimilarity()
id:ID
language:Language code
dataType:Enumeration
contains
1
1
or
0..1
Atomic Descriptor Value

Fig. 3

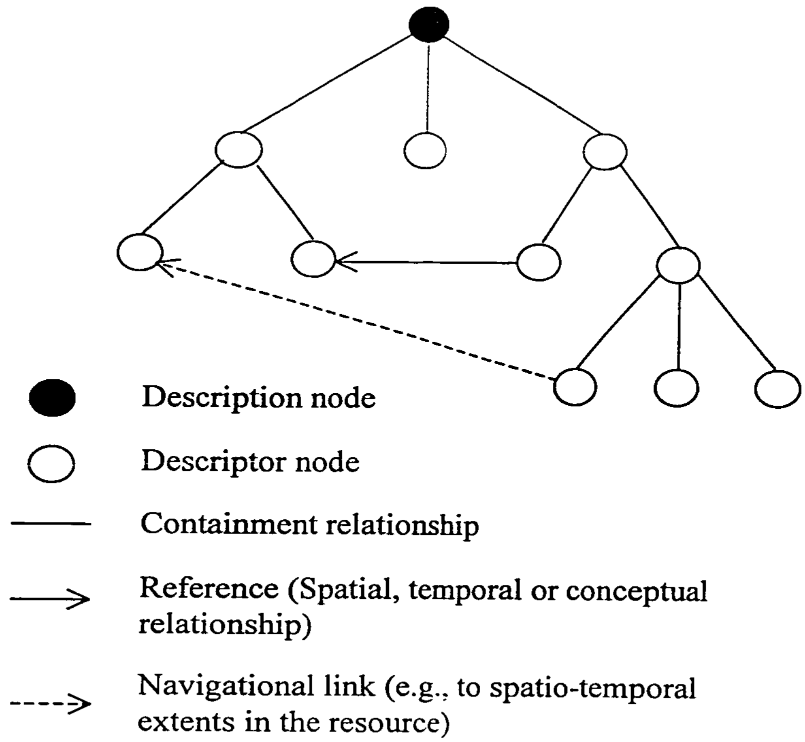
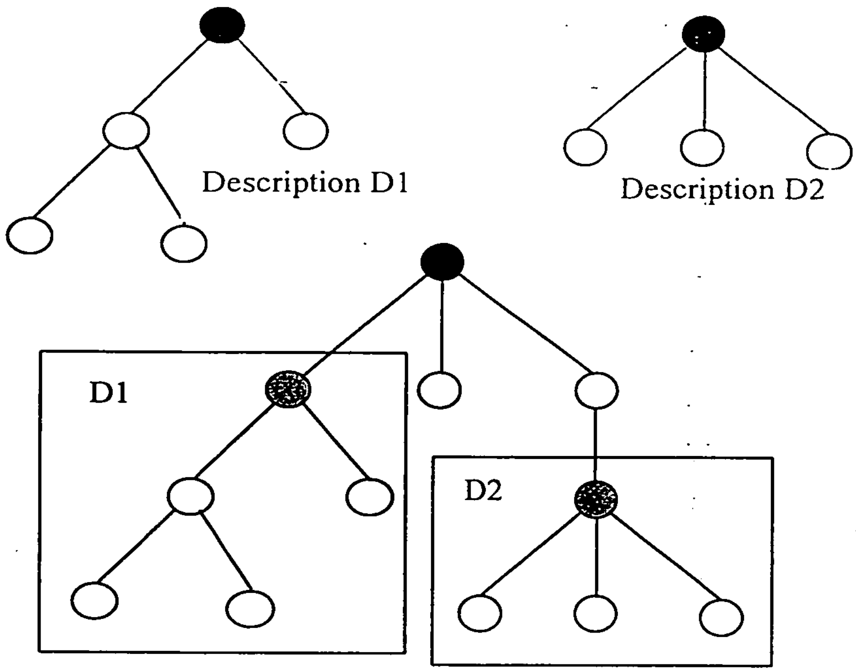

Description node
Descriptor node
Containment relationship
Reference (Spatial, temporal or conceptual relationship)
Navigational link (e.g., to spatio-temporal extents in the resource)

Fig. 4

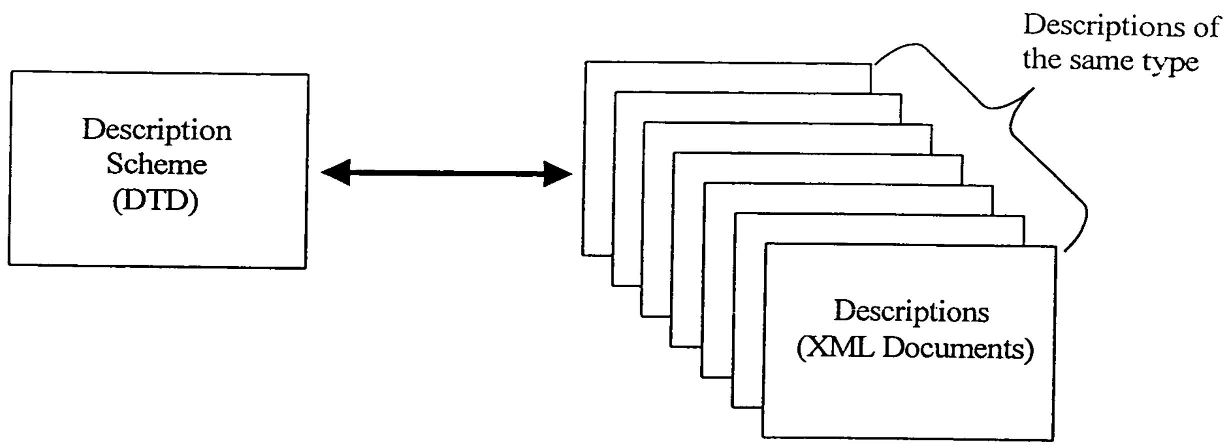


Description D3 containing descriptions D1 and D2 as sub-trees

Fig. 5

6/16

Fig. 6



Description
Scheme
(DTD)
Descriptions of
the same type
Descriptions
(XML Documents)

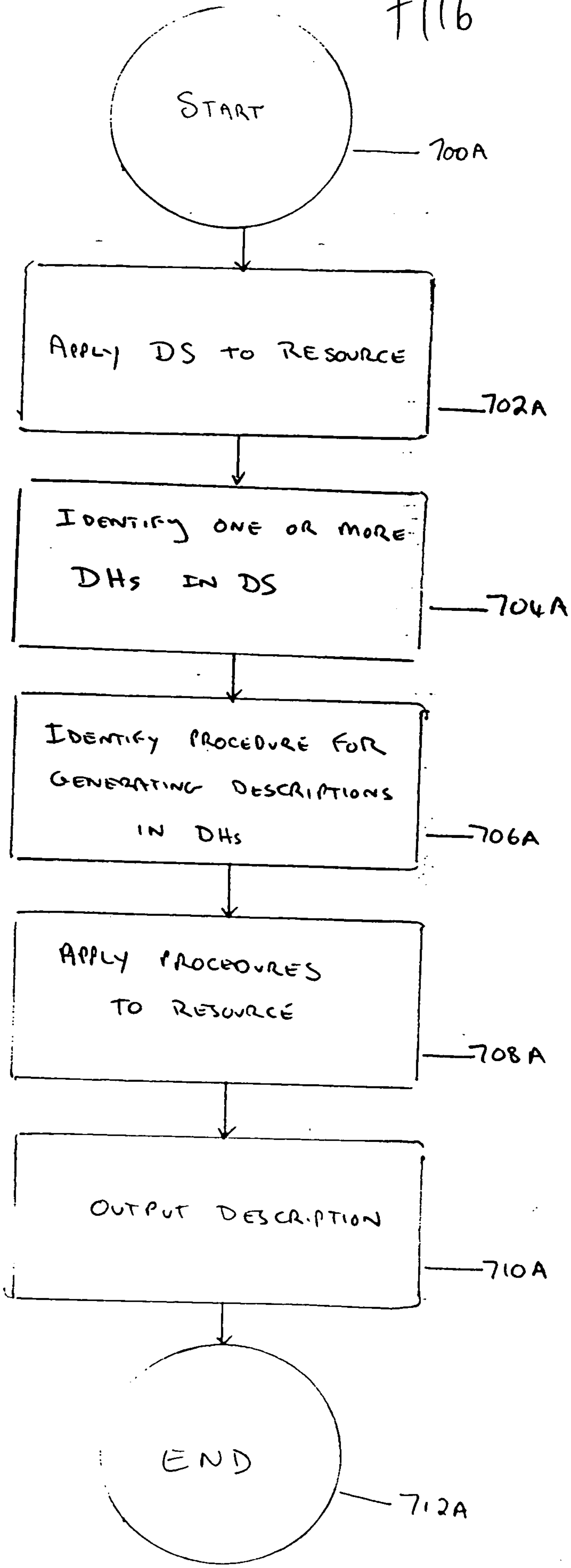

START
700A
APPLY DS TO RESOURCE
702A
IDENTIFY ONE OR MORE DHs IN DS
704A
IDENTIFY PROCEDURE FOR GENERATING DESCRIPTIONS IN DHs
706A
APPLY PROCEDURES TO RESOURCE
708A
OUTPUT DESCRIPTION
710A
END
712A

Fig. 7A

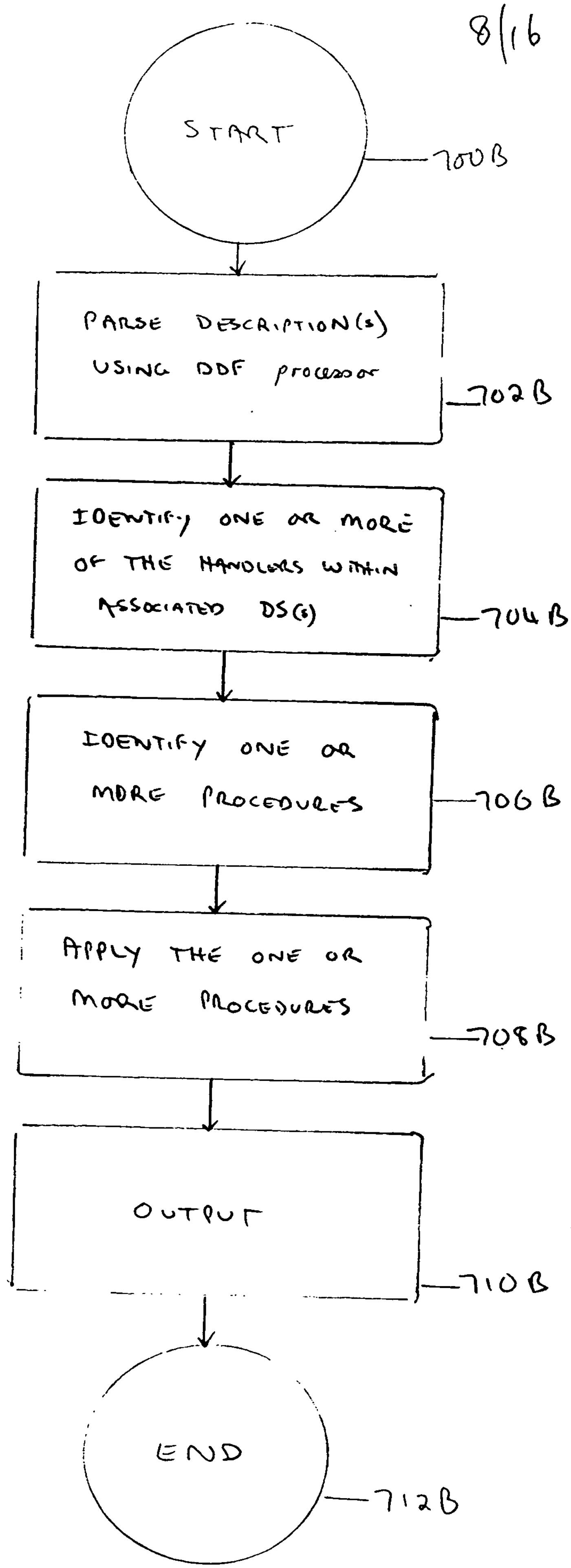

START
700B
PARSE DESCRIPTION(S) USING DDF processor
702B
IDENTIFY ONE OR MORE OF THE HANDLERS WITHIN ASSOCIATED DS(S)
704B
IDENTIFY ONE OR MORE PROCEDURES
706B
APPLY THE ONE OR MORE PROCEDURES
708B
OUTPUT
710B
END
712B

Fig. 7B

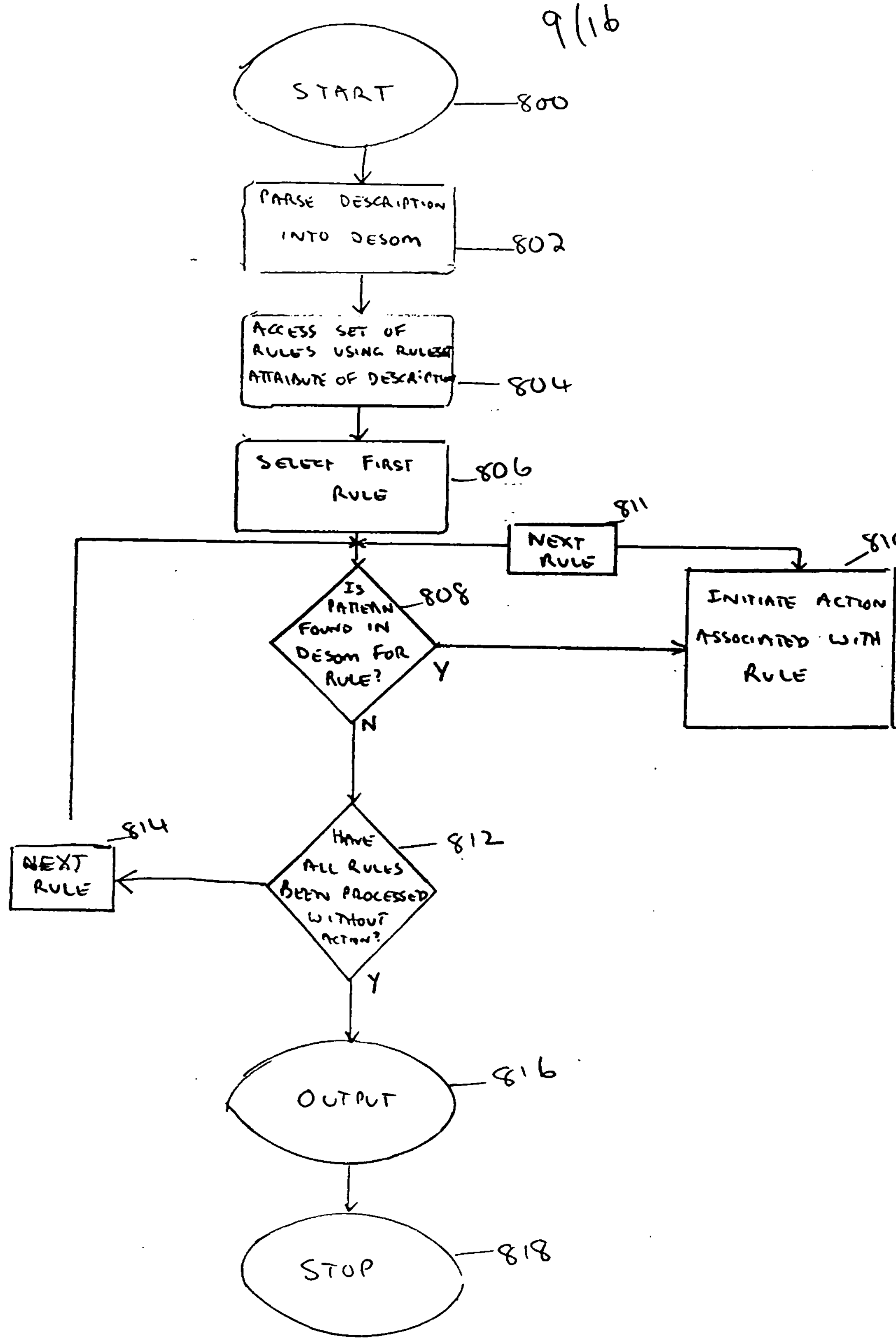

START
800
PARSE DESCRIPTION INTO DESOM
802
ACCESS SET OF RULES USING RULESET ATTRIBUTE OF DESCRIPTION
804
SELECT FIRST RULE
806
NEXT RULE
811
INITIATE ACTION ASSOCIATED WITH RULE
810
IS PATTERN FOUND IN DESOM FOR RULE?
808
Y
N
HAVE ALL RULES BEEN PROCESSED WITHOUT ACTION?
812
NEXT RULE
814
Y
OUTPUT
816
STOP
818

FIG. 8

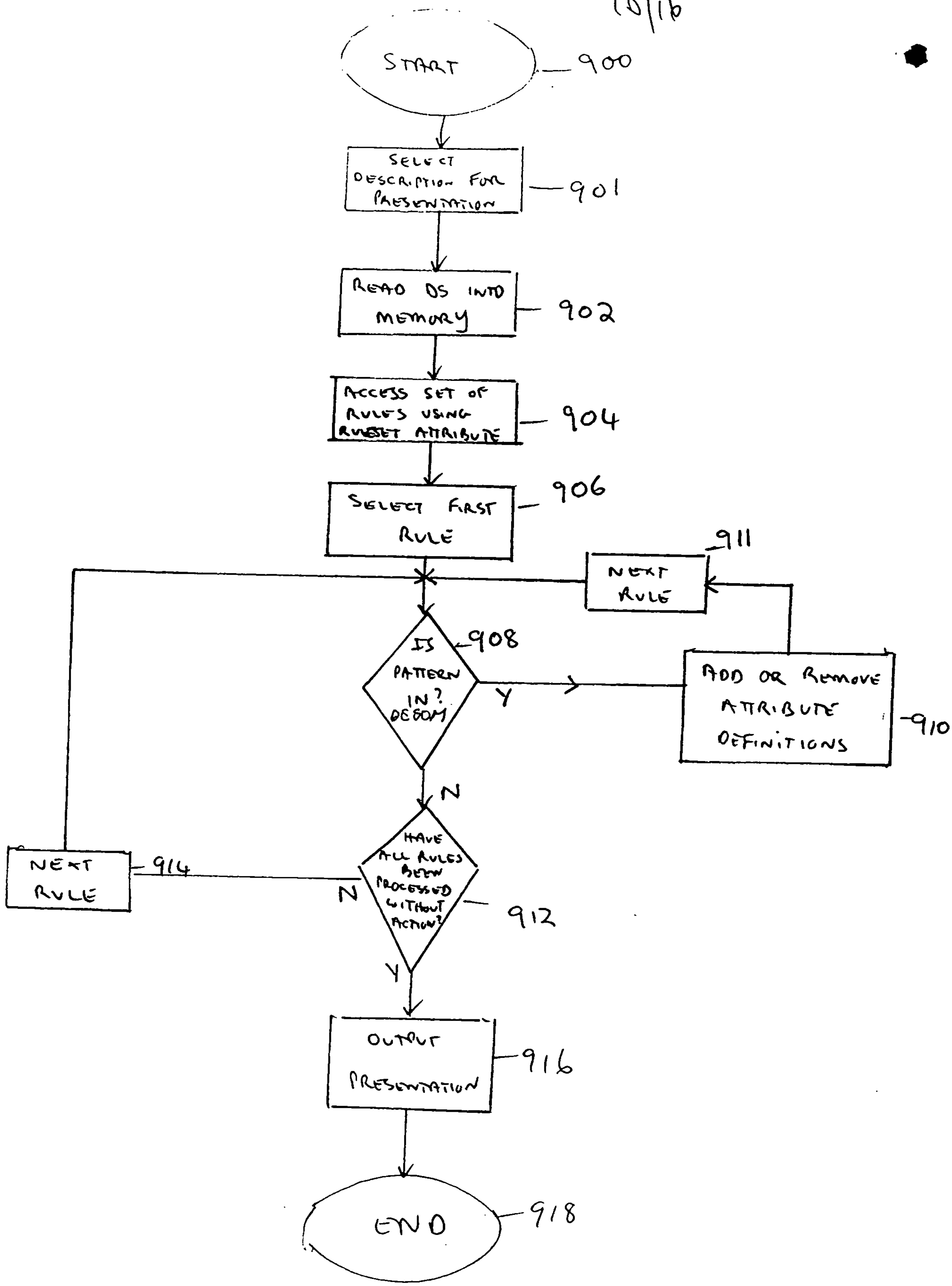

START
900
SELECT DESCRIPTION FOR PRESENTATION
901
READ DS INTO MEMORY
902
ACCESS SET OF RULES USING RULESET ATTRIBUTE
904
SELECT FIRST RULE
906
NEXT RULE
911
IS PATTERN IN DESOM?
908
Y
ADD OR REMOVE ATTRIBUTE DEFINITIONS
910
N
HAVE ALL RULES BEEN PROCESSED WITHOUT ACTION?
912
N
NEXT RULE
914
Y
OUTPUT PRESENTATION
916
END
918

Fig. 9

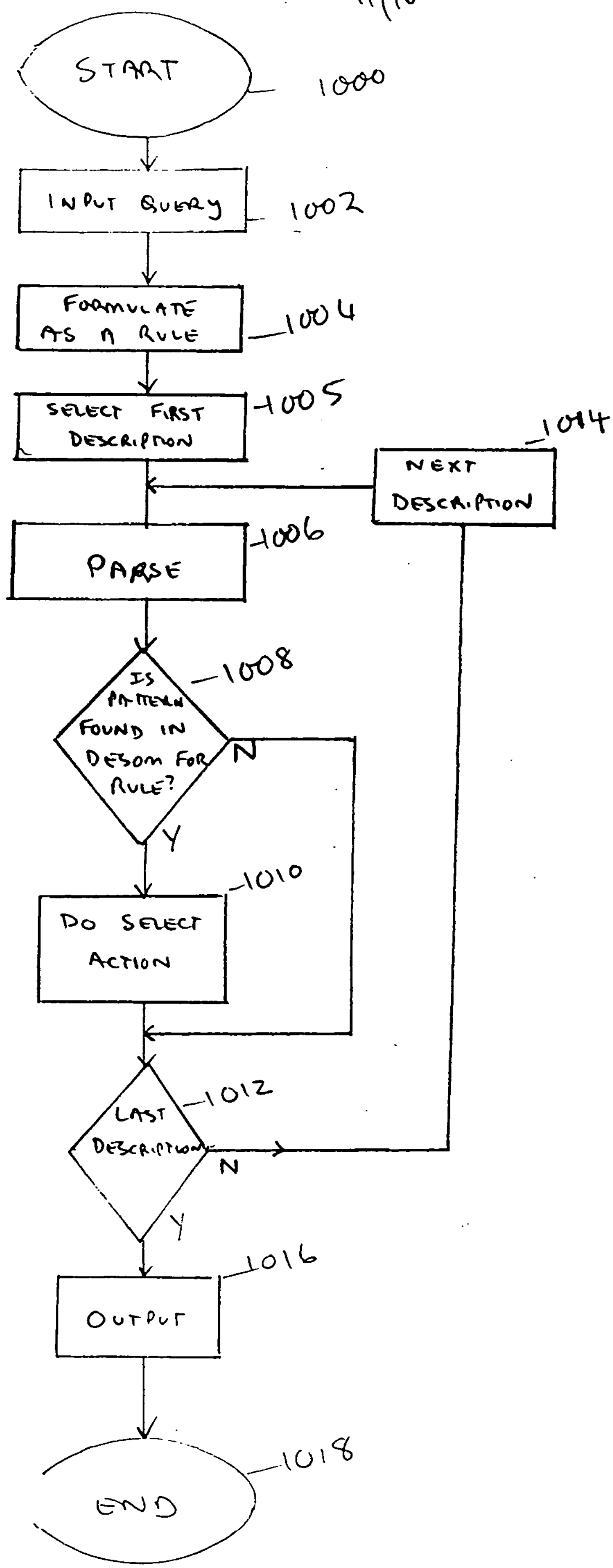

START
1000
INPUT QUERY
1002
FORMULATE AS A RULE
1004
SELECT FIRST DESCRIPTION
1005
NEXT DESCRIPTION
1014
PARSE
1006
IS PATTERN FOUND IN DESCRIPTION FOR RULE?
1008
N
Y
DO SELECT ACTION
1010
LAST DESCRIPTION
1012
N
Y
OUTPUT
1016
END
1018

Fig. 10

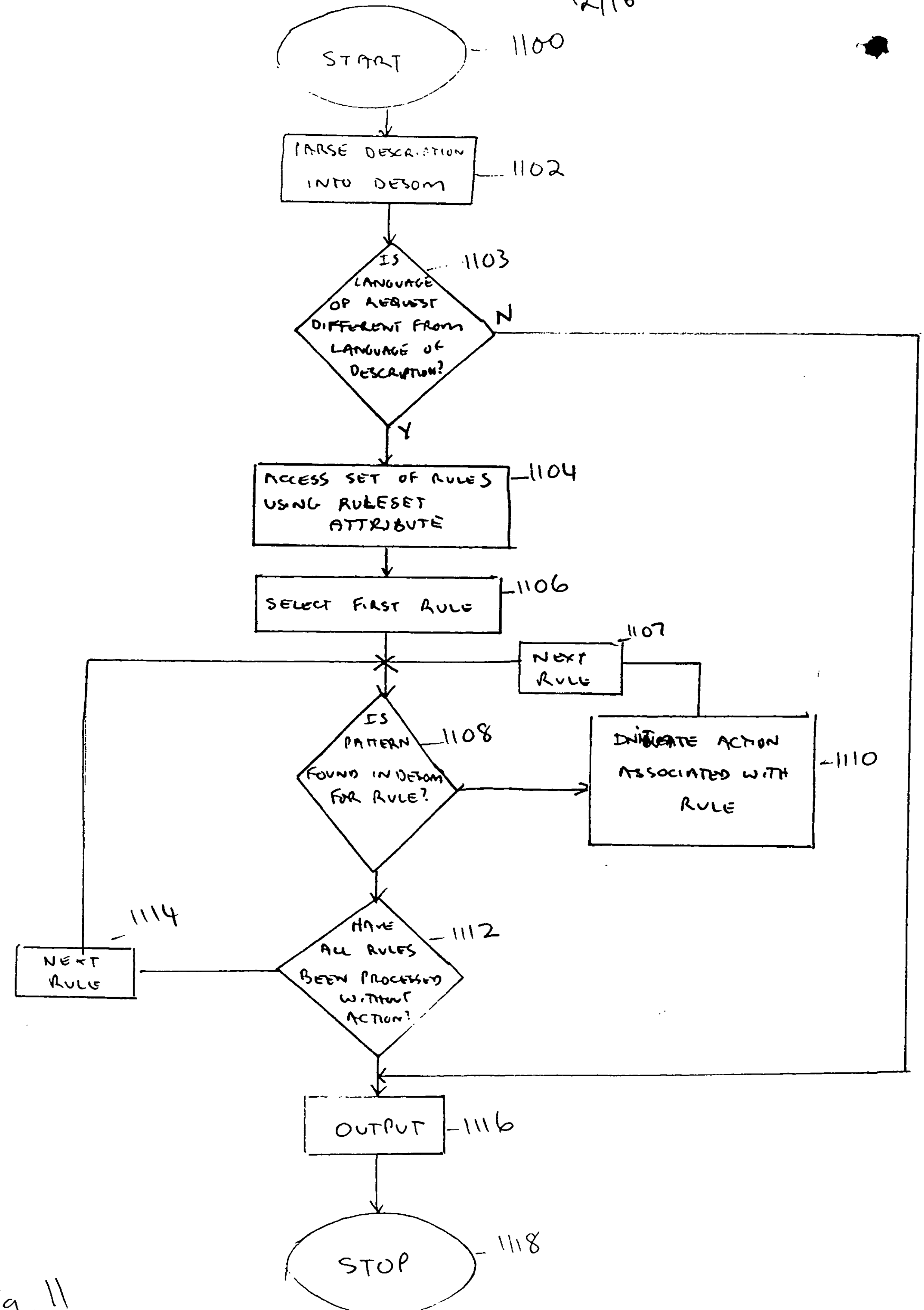

START
1100
PARSE DESCRIPTION INTO DESOM
1102
IS LANGUAGE OF REQUEST DIFFERENT FROM LANGUAGE OF DESCRIPTION?
1103
N
Y
ACCESS SET OF RULES USING RULESET ATTRIBUTE
1104
SELECT FIRST RULE
1106
NEXT RULE
1107
IS PATTERN FOUND IN DESOM FOR RULE?
1108
INITIATE ACTION ASSOCIATED WITH RULE
1110
HAVE ALL RULES BEEN PROCESSED WITHOUT ACTION?
1112
NEXT RULE
1114
OUTPUT
1116
STOP
1118

Fig. 11

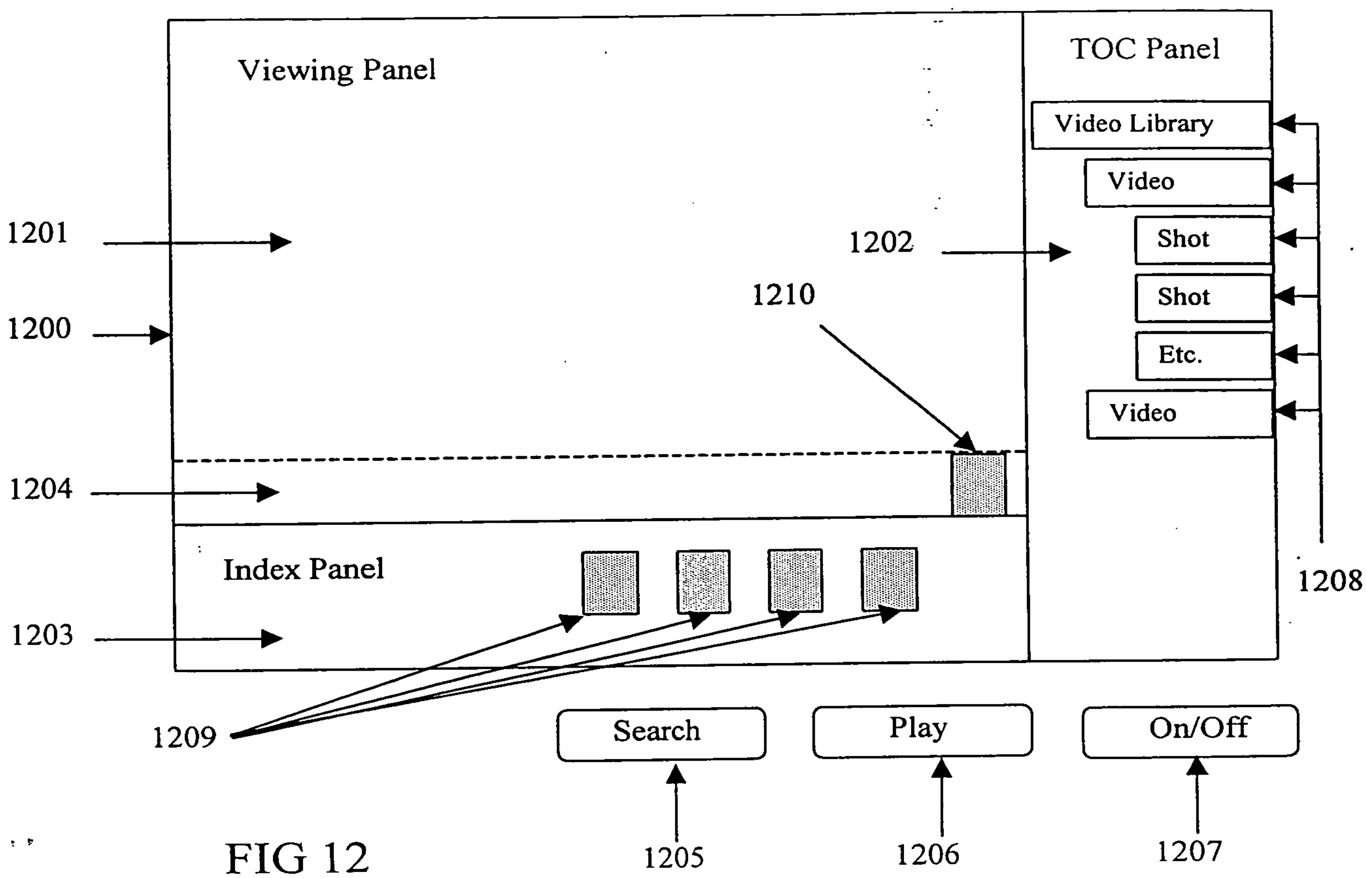

Viewing Panel
TOC Panel
Video Library
Video
Shot
Shot
Etc.
Video
Index Panel
Search
Play
On/Off
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210

FIG 12

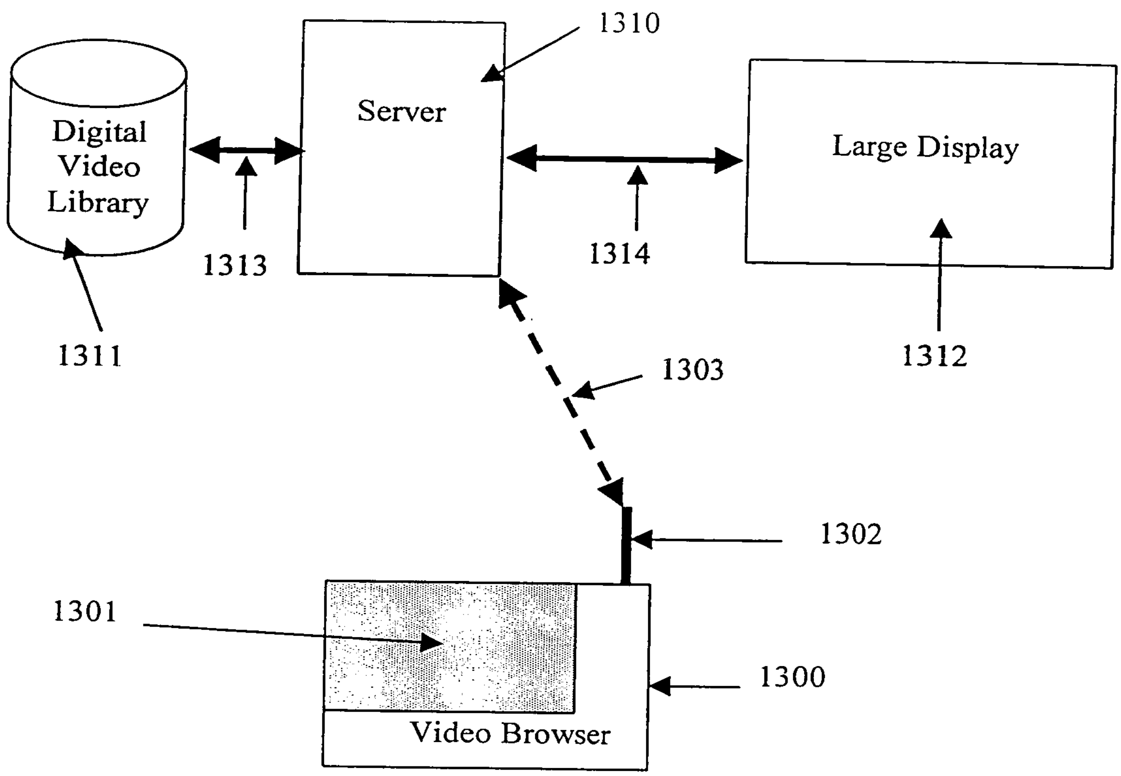

1310
Server
Digital
Video
Library
Large Display
1313
1314
1311
1312
1303
1302
1301
1300
Video Browser

Fig. 13

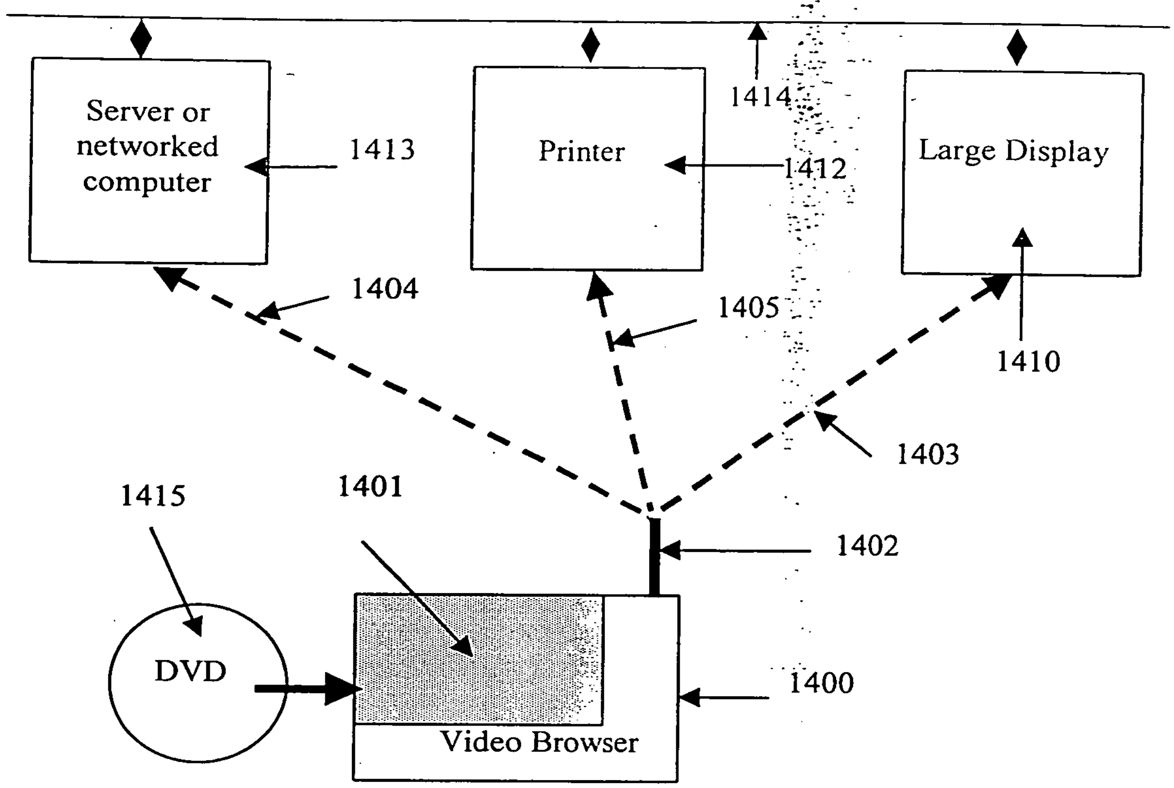

Server or networked computer
1413
Printer
1414
1412
Large Display
1404
1405
1410
1403
1415
1401
1402
DVD
1400
Video Browser

Fig. 14

16/16

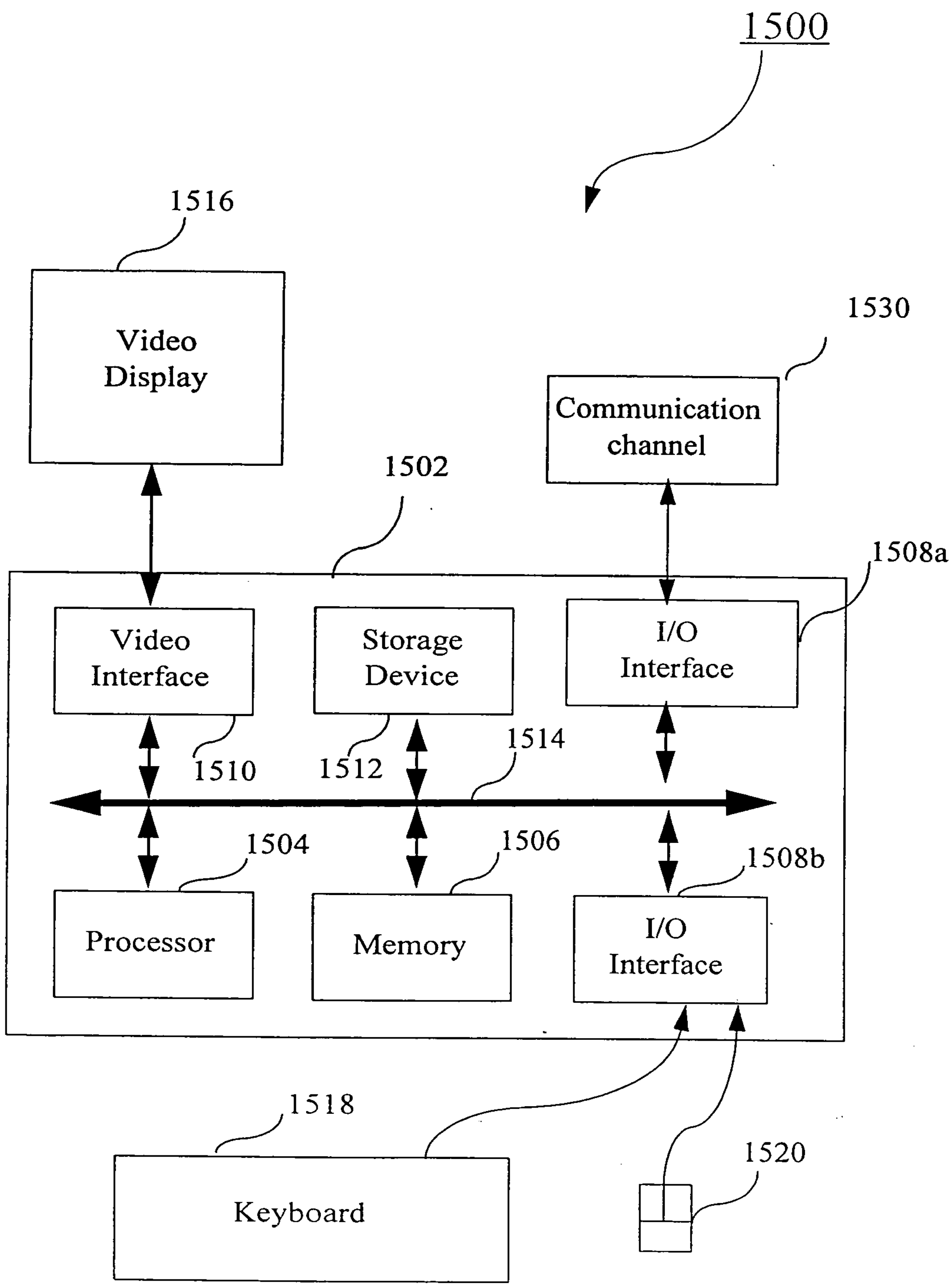


Fig 15

FITZPATRICK, CELLA, HARPER & SCINTO
30 Rockefeller Plaza
New Y rk, NY 10112-3801